BET's

BASAVAKALYAN ENGINEERING COLLEGE

BASAVAKALYAN

Department

Of

Computer Science & Engineering

COMPUTER NETWORK LABORATORY

(18CSL57)

V Semester (CBCS)

Prepared by:

Mr.Gangadhara GH

Associate Professor

COMPUTER NETWORK LABORATORY

[As per Choice Based Credit System (CBCS) scheme]
(Effective from the academic year 2018 -2019)
SEMESTER – V

Subject Code: 18CSL57 **CIE Marks: 40**

Number of Contact Hours/Week 0:2:2 **SEE Marks: 60**

Total Number of Lab Contact Hours: 36 **Exam Hours: 3 Hrs**

Credits – 2

Course Learning Objectives: This course (18CSL57) will enable students to:

- Demonstrate operation of network and its management commands
- Simulate and demonstrate the performance of GSM and CDMA
- Implement data link layer and transport layer protocols.

Descriptions (if any):

- For the experiments below modify the topology and parameters set for the experiment and take multiple rounds of reading and analyze the results available in log files. Plot necessary graphs and conclude. Use NS2/NS3.
- Installation procedure of the required software must be demonstrated, carried out in groups and documented in the journal.

Programs List:

PART A

1. Implement three nodes point – to – point network with duplex links between them. Set the queue size, vary the bandwidth and find the number of packets dropped.
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.
3. Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.
4. Implement simple ESS and with transmitting nodes in wire-less LAN by simulation and determine the performance with respect to transmission of packets.
5. Implement and study the performance of GSM on NS2/NS3 (Using MAC layer) or equivalent environment.
6. Implement and study the performance of CDMA on NS2/NS3 (Using stack called Call net) or equivalent environment

PART B (Implement the following in Java)

7. Write a program for error detecting code using CRC-CCITT (16- bits).
8. Write a program to find the shortest path between vertices using bellman-ford algorithm.
9. 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.
10. Write a program on datagram socket for client/server to display the messages on client side, typed at the server side.
11. Write a program for simple RSA algorithm to encrypt and decrypt the data.
12. Write a program for congestion control using leaky bucket algorithm.

Laboratory Outcomes: The student should be able to:

- Analyze and Compare various networking protocols.
- Demonstrate the working of different concepts of networking.
- Implement, analyze and evaluate networking protocols in NS2 / NS3 and JAVA programming language

Conduct of Practical Examination:

• Experiment distribution:

  - For laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.
  - For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.

- Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.
- Marks Distribution (Subjected to change in accoradance with university regulations)
  a) For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks
  b) For laboratories having PART A and PART B
     i. Part A – Procedure + Execution + Viva = 6 + 28 + 6 = 40 Marks
     ii. Part B – Procedure + Execution + Viva = 9 + 42 + 9 = 60 Marks

Part A - SIMULATION USING NS-2

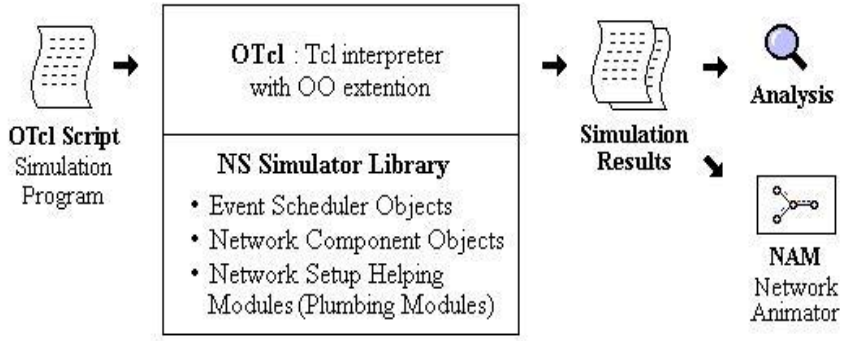
Introduction to NS-2:

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

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,MACetc
- Error Models: Uniform, burstyetc
- 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 unixsystems(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.

Protocols implemented in NS2

Transport layer(Traffic Agent) – TCP, UDP

Network layer(Routing agent)

Interface queue – FIFO queue, Drop Tail queue, Priority queue

Logic link contol layer – IEEE 802.2, AR

How to use NS2

Design Simulation – Determine simulation scenario

Build ns-2 script using tcl.

Run simulation

Simulation with NS2

Define objects of simulation.

Connect the objects to each other Start the source applications.

Packets are then created and are transmitted through network.

Exit the simulator after a certain fixed time.

NS programming Structure

- Create the event scheduler
- Turn on tracing
- Create network topology
- Create transport connections
- Generate traffic
- Insert errors

Sample Wired Simulation using NS-2

Creating Event Scheduler

- Create event scheduler: set ns [new simulator]
- Schedule an event: $ns at <time><event>

  – event is any legitimate ns/tcl function

  $ns at 5.0 “finish”

  proc finish {} {

  global ns nf

  close $nf

  exec namout.nam&

  exit 0

  }

- Start Scheduler

  $ns run

Tracing

- All packet trace

  $ns traceall[open out.tr w]

<event><time><from><to><pkt><size>

  …

  <flowid><src><dst><seqno><aseqno>

  + 0.51 0 1 cbr 500 —— 0 0.0 1.0 0 2

_ 0.51 0 1 cbr 500 —– 0 0.0 1.0 0 2

R 0.514 0 1 cbr 500 —–

0 0.0 1.0 0 0

- Variable trace

set par [open output/param.tr w]

$tcp attach $par

$tcp trace cwnd_

$tcp trace maxseq_

$tcp trace rtt_

Tracing and Animation

- Network Animator

set nf [open out.nam w]

$ns namtraceall

$nf

proc finish {} {

global ns nf

close $nf

exec namout.nam&

exit 0

}

Creating topology

- Two nodes connected by a link
- - Creating nodes

set n0 [$ns node]

set n1 [$ns node]

- Creating link between nodes

$ns <link_type> $n0 $n1 <bandwidth><delay><queue-type> $ns duplex-link$n0 $n1 1Mb 10ms DropTail

Data Sending

- Create UDP agent

set udp0 [new Agent/UDP]

$ns attach-agent $n0 $udp0

- Create CBR traffic source for feeding into UDP

agent set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize_

500 $cbr0 set interval_

0.005 $cbr0 attach-

agent$udp0

- Create traffic sink

set null0 [new Agent/Null]

$ns attach-agent$n1 $null0

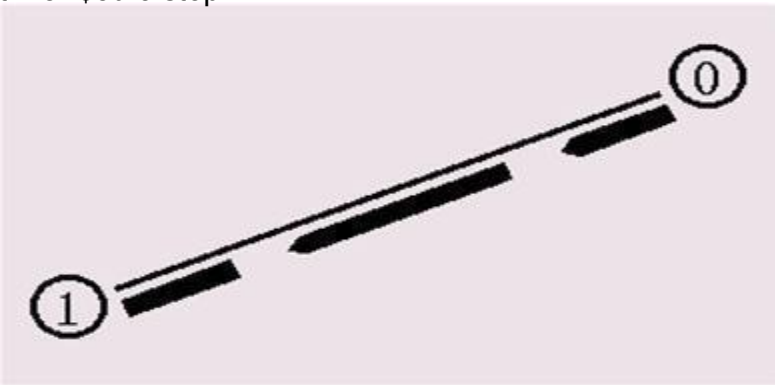
- Connect two agents

$ns connect $udp0 $null0

- Start and stop of data

$ns at 0.5 "$cbr0 start"

$ns at 4.5 "$cbr0 stop"



Traffic on top of TCP

- FTP

set ftp [new Application/FTP]

$ftp attach-agent$tcp0

- Telnet

set telnet [new Application/Telnet]

$telnet attach-agent$tcp0

PROCEDURE

STEP 1: Start

STEP 2: Create the simulator object ns for designing the given simulation

STEP 3: Open the trace file and nam file in the write mode

STEP 4: Create the nodes of the simulation using the 'set' command

STEP 5: Create links to the appropriate nodes using $ns duplex-link command

STEP 6: Set the orientation for the nodes in the simulation using 'orient' command

STEP 7: Create TCP agent for the nodes and attach these agents to the nodes

STEP 8: The traffic generator used is FTP for both node0 and node1

STEP 9: Configure node1 as the sink and attach it

STEP10: Connect node0 and node1 using 'connect' command

STEP 11: Setting color for the nodes

STEP 12: Schedule the events for FTP agent 10 sec

STEP 13: Schedule the simulation for 5 minutes

Structure of Trace Files

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

The meaning of the fields are:

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

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.

1. The second field gives the time at which the event occurs.
2. Gives the input node of the link at which the event occurs.
3. Gives the output node of the link at which the event occurs.
4. Gives the packet type (eg CBR or TCP)
5. Gives the packet size
6. Some flags
7. 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.
8. This is the source address given in the form of –node.port‖.
9. This is the destination address, given in the same form.
10. 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
11. 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‖.

Part A

EXPERIMENT 1

Simulate a three-node point-to-point network with a duplex link between them. Set the queue size and vary the bandwidth and find the number of packets dropped.

Program:

```
set ns [new Simulator]                          # Letter S is capital
set nf [open PA1.nam w]                         # open a nam trace file in write mode
$ns namtrace-all $nf                            # nfnam filename
set tf [open PA1.tr w]                          # tf  trace filename
$ns trace-all $tf

proc finish { } {
        global ns nftf
        $ns flush-trace                         # clears trace file contents
        close $nf
        close $tf
        exec nam PA1.nam &
        exit 0
}
set n0 [$ns node]                               # creates 3 nodes
set n2 [$ns node]
set n3 [$ns node]

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

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

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

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

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

```
#immediately after BEGIN should open braces '{'
BEGIN{ c=0;}
{
if($1= ="d")
```

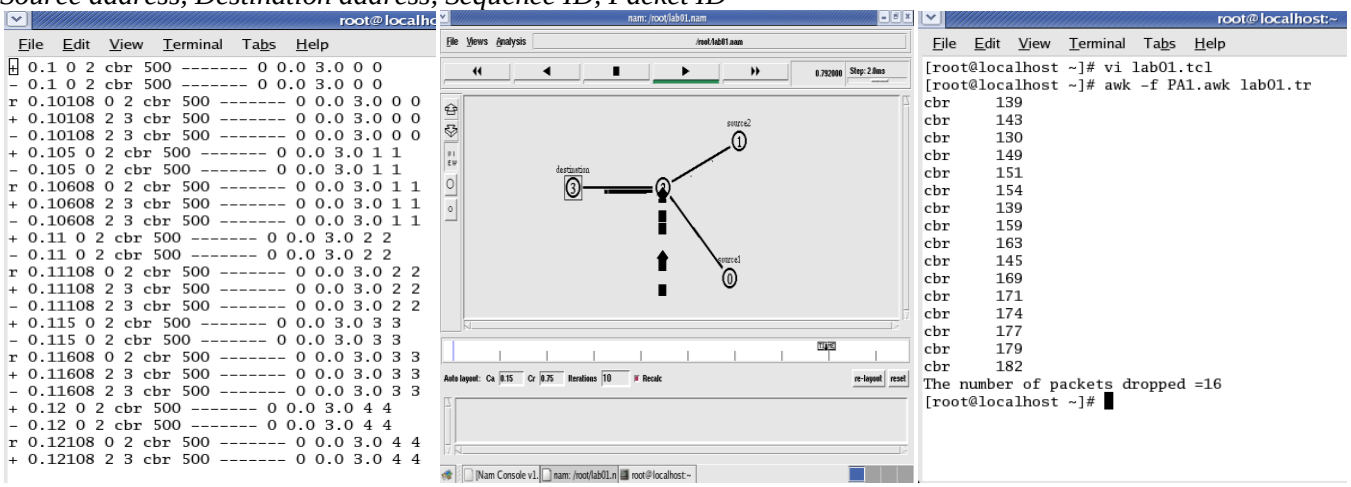
```
 {      c++;
        printf("%s\t%s\n",$5,$11);
 }
}
END{ printf("The number of packets dropped =%d\n",c); }
```

Steps for execution:

- *Open vi editor and type program. Program name should have the extension " **.tcl** "*
  [root@localhost~]# vi lab1.tcl
- *Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.*
- *Open vi editor and type **awk** program. Program name should have the extension "**.awk** "*
  [root@localhost~]# vi lab1.awk
- *Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.*
- *Run the simulation program*
  [root@localhost~]# ns lab1.tcl
- *Here **"ns"** indicates network simulator. We get the topology shown in the snapshot.*
- *Now press the play button in the simulation window and the simulation will begins.*
- *After simulation is completed run **awk file** to see the output ,*
  [root@localhost~]#awk –f lab1.awk lab1.tr
- *To see the trace file contents open the file as ,*
  [root@localhost~]# vi lab1.tr

Trace file contains 12 columns:

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



Contents of Trace File Topology Output

EXPERIMENT 2

Simulate the transmission of ping messages over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.

Program:

```
set ns [ new Simulator ]
set nf [ open lab4.nam w ]
$ns namtrace-all $nf
set tf [ open lab4.tr w ]
$ns trace-all $tf
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

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

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

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

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


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

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

$ns queue-limit $n0 $n4 5
$ns queue-limit $n2 $n4 3
$ns queue-limit $n4 $n5 2

Agent/Ping instprocrecv {from rtt} {
```

```
$self instvar node_
puts "node [$nod e_ id]received answer from $from with round trip time $rttmsec"
}
# please provide space between $node_ and id. No space between $ and from. No space between and $
and rtt */

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

proc finish { } {
global ns nftf
$ns flush-trace
close $nf
close $tf
exec nam lab4.nam &
exit 0
}
$ns at 0.1 "$p1 send"
$ns at 0.2 "$p1 send"
$ns at 0.3 "$p1 send"
$ns at 0.4 "$p1 send"
$ns at 0.5 "$p1 send"
$ns at 0.6 "$p1 send"
$ns at 0.7 "$p1 send"
$ns at 0.8 "$p1 send"
$ns at 0.9 "$p1 send"
$ns at 1.0 "$p1 send"
$ns at 1.1 "$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
```

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

```
BEGIN{
drop=0;
}
{
if($1= ="d" )
  {
   drop++;
   }
}
END{
printf("Total number of %s packets dropped due to congestion =%d\n",$5,drop);
}
```

Steps for execution

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

   [root@localhost ~]# vi lab4.tcl

2) *Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.*

3) *Open vi editor and type **awk** program. Program name should have the extension "**.awk** "*

   [root@localhost~]# vi lab4.awk

4) *Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.*

5) *Run the simulation program*

   [root@localhost~]# ns lab4.tcl

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

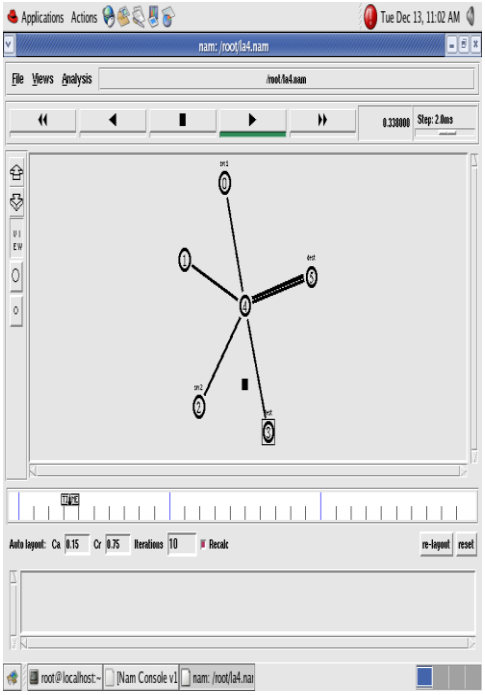
   ii) *Now press the play button in the simulation window and the simulation will begins.*

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

   [root@localhost~]#awk –f lab4.awk lab4.tr

7) *To see the trace file contents open the file as ,*

   [root@localhost~]# vi lab4.tr



Topology

Output

EXPERIMENT 3

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

Program:

```
set ns [new Simulator]
set tf [open pgm7.tr w]
$ns trace-all $tf
set nf [open pgm7.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  #
should come in single line
$ns duplex-link $n4 $n5 1Mb 1ms DropTail

set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ftp0 set packetSize_ 500
$ftp0 set interval_ 0.0001
set sink5 [new Agent/TCPSink]
$ns attach-agent $n5 $sink5

$ns connect $tcp0 $sink5

set tcp2 [new Agent/TCP]
$ns attach-agent $n2 $tcp2
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ftp2 set packetSize_ 600
$ftp2 set interval_ 0.001
```

```
set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3

$ns connect $tcp2 $sink3

set file1 [open file1.tr w]
$tcp0 attach $file1
set file2 [open file2.tr w]
$tcp2 attach $file2

$tcp0 trace cwnd_  # must put underscore ( _ ) after cwnd and no space between them
$tcp2 trace cwnd_

proc finish { } {
global ns nftf
$ns flush-trace
close $tf
close $nf
exec nam pgm7.nam &
exit 0
}

$ns at 0.1 "$ftp0 start"
$ns at 5 "$ftp0 stop"
$ns at 7 "$ftp0 start"
$ns at 0.2 "$ftp2 start"
$ns at 8 "$ftp2 stop"
$ns at 14 "$ftp0 stop"
$ns at 10 "$ftp2 start"
$ns at 15 "$ftp2 stop"
$ns at 16 "finish"
$ns run
```

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

cwnd:- means congestion window

```
BEGIN {

}

{

if($6= ="cwnd_")  # don't leave space after writing cwnd_

printf("%f\t%f\t\n",$1,$7); # you must put \n in printf

}

END {

}
```

Steps for execution:

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

   [root@localhost ~]# vi lab7.tcl

2) *Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.*

3) *Open vi editor and type **awk** program. Program name should have the extension "**.awk** "*

   [root@localhost~]# vi lab7.awk

4) *Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.*

5) *Run the simulation program*

   [root@localhost~]# ns lab7.tcl

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

   i. ***[root@localhost~]# awk –f lab7.awk file1.tr >a1***

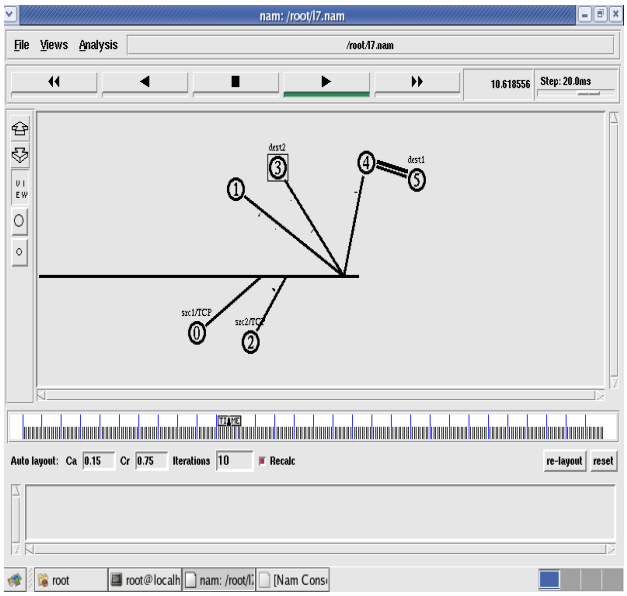
   ii. ***[root@localhost~]# awk –f lab7.awk file2.tr >a2***

   iii. ***[root@localhost~]# xgraph a1 a2***

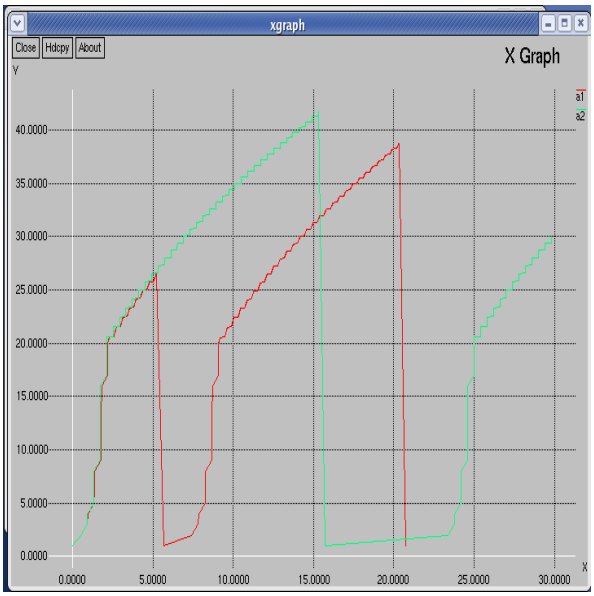
7) *Here we are using the congestion window trace files i.e. **file1.tr** and **file2.tr** and we are redirecting the contents of those files to new files say **a1** and **a2** using **output redirection operator (>)**.*

8) *To see the trace file contents open the file as ,*

   [root@localhost~]# vi lab7.tr



Output



Graph

EXPERIMENT 4

Simulate simple ESS and with transmitting nodes in wireless LAN by simulation and determine the performance with respect to transmission of packets.

Program:

```
set ns [new Simulator]
set tf [open lab8.tr w]
$ns trace-all $tf
set topo [new Topography]
$topo load_flatgrid 1000 1000
set nf [open lab8.nam w]
$ns namtrace-all-wireless $nf 1000 1000

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

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

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

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

$ns at 0.1 "$n0 setdest 50 50 15"
$ns at 0.1 "$n1 setdest 100 100 25"
$ns at 0.1 "$n2 setdest 600 600 25"
 set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
set sink1 [new Agent/TCPSink]
$ns attach-agent $n1 $sink1
$ns connect $tcp0 $sink1
set tcp1 [new Agent/TCP]
```

```
$ns attach-agent $n1 $tcp1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
set sink2 [new Agent/TCPSink]
$ns attach-agent $n2 $sink2
$ns connect $tcp1 $sink2
$ns at 5 "$ftp0 start"
$ns at 5 "$ftp1 start"
$ns at 100 "$n1 setdest 550 550 15"
$ns at 190 "$n1 setdest 70 70 15"
proc finish { } {
        global ns nftf
        $ns flush-trace
        exec namlab8.nam  &
        close $tf
        exit 0
}
$ns at 250 "finish"
$ns run
```

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

```
BEGIN{
        count1=0
        count2=0
        pack1=0
        pack2=0
        time1=0
        time2=0
}
{       if($1= ="r"&& $3= ="_1_" && $4= ="AGT")
        {       count1++
                pack1=pack1+$8
                time1=$2       }
        if($1= ="r" && $3= ="_2_" && $4= ="AGT")
        {       count2++
                pack2=pack2+$8
                time2=$2       }
}
END{
printf("The Throughput from n0 to n1: %f Mbps \n", ((count1*pack1*8)/(time1*1000000)));
printf("The Throughput from n1 to n2: %f Mbps", ((count2*pack2*8)/(time2*1000000)));
}
```

Steps for execution

1) Open vi editor and type program. Program name should have the extension **— .tcl**

[root@localhost~]# vi lab4.tcl

2) Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.

3) Open vi editor and type **awk**program. Program name should have the extension **—.awk**

[root@localhost~]# vi lab4.awk

4) Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys

simultaneously and type **"wq"** and press **Enter key**.

5) Run the simulation program

[root@localhost~]# ns lab4.tcl

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

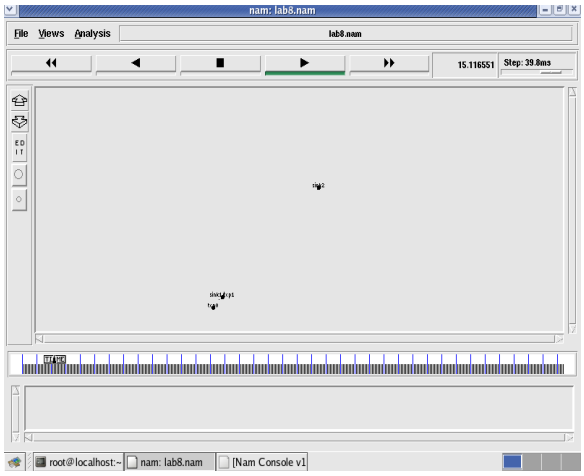
ii) Now press the play button in the simulation window and the simulation will begins.

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

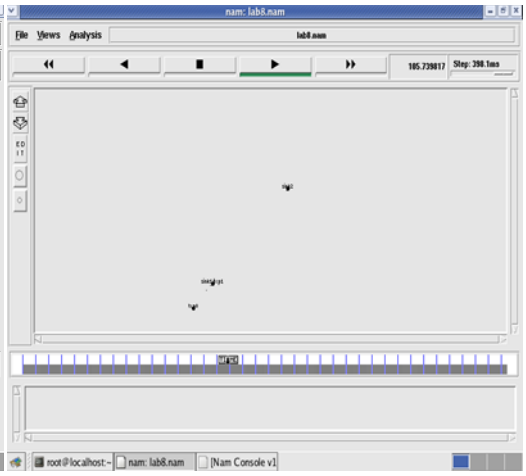
[root@localhost~]#awk–f lab4.awk lab4.tr

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

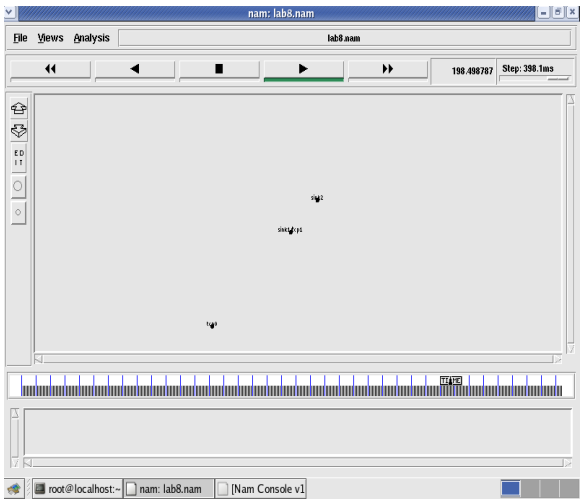
[root@localhost~]# vi lab4.trTopology:

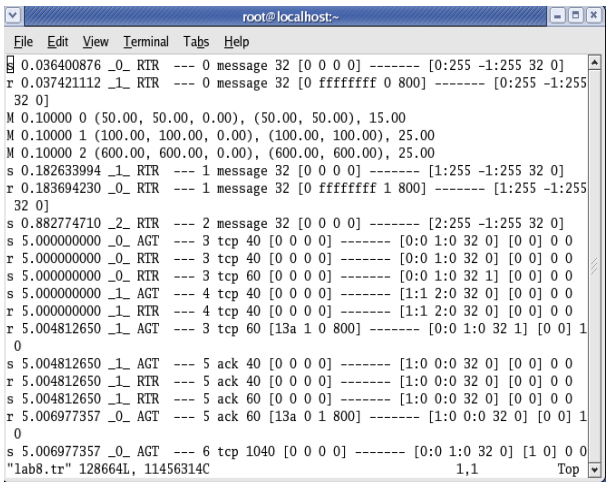


Node 1 and 2 are communicating



Node 2 is moving towards node 3





Node 2 is coming back from node 3 towards node1Trace File

EXPERIMENT 5

Implement and study the performance of GSM on NS2/NS3 (Using MAC layer) or equivalent environment.

Second Generation (2G) technology is based on the technology known as global system for mobile communication (GSM). This technology enabled various networks to provide services like text messages, picture messages and MMS. The technologies used in 2G are either TDMA (Time Division Multiple Access) which divides signal into different time slots or CDMA (Code Division Multiple Access) which allocates a special code to each user so as to communicate over a multiplex physical channel.

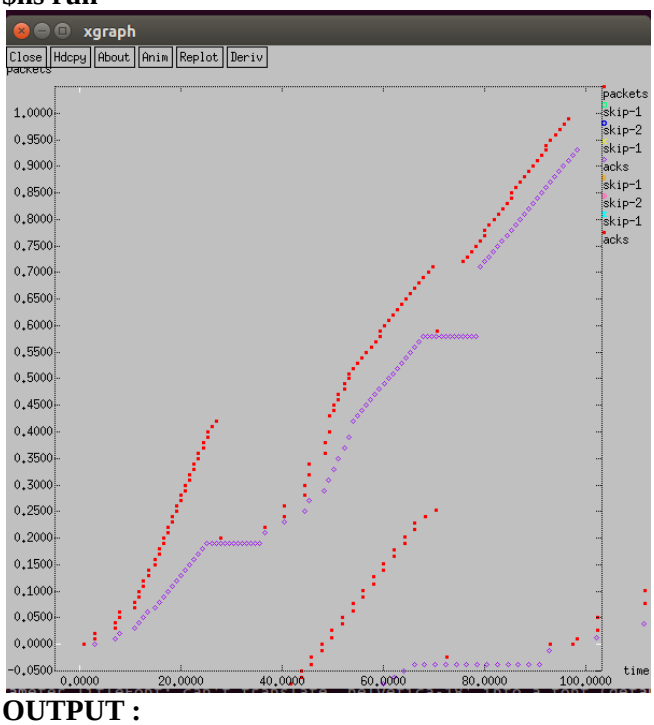
GSM uses a variation of time division multiple access (TDMA). 2G networks developed as a replacement for first generation (1G) analog cellular networks, and the GSM standard originally described as a digital, circuit-switched network optimized for full duplex voice telephony. This expanded over time to include data communications, first by circuit switched transport, then by packet data transport via GPRS (General Packet Radio Services). GSM can be implemented on all the versions of NS2 (Since year 2004: ns-2.27, and later versions of NS2)

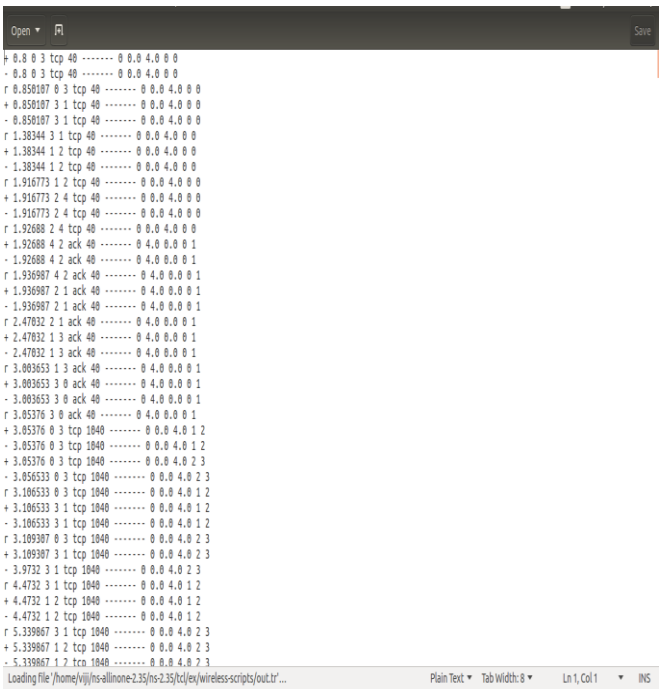
Program:

```
# General Parameters
set opt(title) zero ;
set opt(stop) 100 ;# Stop time.
set opt(ecn) 0 ;
# Topology
set opt(type) gsm ;#type of link:
set opt(secondDelay) 55 ;# average delay of access links in ms
# AQM parameters
set opt(minth) 30 ;
set opt(maxth) 0 ;
set opt(adaptive) 1 ;# 1 for Adaptive RED, 0 for plain RED
# Traffic generation.
set opt(flows) 0 ;# number of long-lived TCP flows
set opt(window) 30 ;# window for long-lived traffic
set opt(web) 2 ;# number of web sessions
# Plotting statistics.
set opt(quiet) 0 ;# popup anything
set opt(wrap) 100 ;# wrap plots
set opt(srcTrace) is ;# where to plot traffic
set opt(dstTrace) bs2 ;# where to plot traffic
set opt(gsmbuf) 10 ; # buffer size for gsm
#default downlink bandwidth in bps
set bwDL(gsm) 9600
#default uplink bandwidth in bps
set bwUL(gsm) 9600
#default downlink propagation delay in seconds
set propDL(gsm) .500
#default uplink propagation delay in seconds
set propUL(gsm) .500
#default buffer size in packets
set buf(gsm) 10
set ns [new Simulator]
set tf [open out.tr w]
$ns trace-all $tf
set nodes(is) [$ns node]
set nodes(ms) [$ns node]
set nodes(bs1) [$ns node]

set nodes(bs2) [$ns node]
set nodes(lp) [$ns node]
proc cell_topo {} {
```

```
global ns nodes
$ns duplex-link $nodes(lp) $nodes(bs1) 3Mbps 10ms DropTail
$ns duplex-link $nodes(bs1) $nodes(ms) 1 1 RED
$ns duplex-link $nodes(ms) $nodes(bs2) 1 1 RED
$ns duplex-link $nodes(bs2) $nodes(is) 3Mbps 50ms DropTail
puts "Cell Topology"
}
proc set_link_params {t} {
global ns nodes bwULbwDLpropULpropDLbuf
$ns bandwidth $nodes(bs1) $nodes(ms) $bwDL($t) simplex
$ns bandwidth $nodes(ms) $nodes(bs1) $bwUL($t) simplex
$ns bandwidth $nodes(bs2) $nodes(ms) $bwDL($t) simplex
$ns bandwidth $nodes(ms) $nodes(bs2) $bwUL($t) simplex
$ns delay $nodes(bs1) $nodes(ms) $propDL($t) simplex
$ns delay $nodes(ms) $nodes(bs1) $propDL($t) simplex
$ns delay $nodes(bs2) $nodes(ms) $propDL($t) simplex
$ns delay $nodes(ms) $nodes(bs2) $propDL($t) simplex
$ns queue-limit $nodes(bs1) $nodes(ms) $buf($t)
$ns queue-limit $nodes(ms) $nodes(bs1) $buf($t)
$ns queue-limit $nodes(bs2) $nodes(ms) $buf($t)
$ns queue-limit $nodes(ms) $nodes(bs2) $buf($t)
}
# RED and TCP parameters
Queue/RED set summarystats_ true
Queue/DropTail set summarystats_ true
Queue/RED set adaptive_ $opt(adaptive)
Queue/RED set q_weight_ 0.0
Queue/RED set thresh_ $opt(minth)
Queue/RED set maxthresh_ $opt(maxth)
Queue/DropTail set shrink_drops_ true
Agent/TCP set ecn_ $opt(ecn)
Agent/TCP set window_ $opt(window)
DelayLink set avoidReordering_ true
source web.tcl

#Create topology
switch $opt(type) {
gsm -
gprs -
umts {cell_topo}
}
set_link_params $opt(type)
$ns insert-delayer $nodes(ms) $nodes(bs1) [new Delayer]
$ns insert-delayer $nodes(bs1) $nodes(ms) [new Delayer]
$ns insert-delayer $nodes(ms) $nodes(bs2) [new Delayer]
$ns insert-delayer $nodes(bs2) $nodes(ms) [new Delayer]
# Set up forward TCP connection
if {$opt(flows) == 0} {
set tcp1 [$ns create-connection TCP/Sack1 $nodes(is) TCPSink/Sack1 $nodes(lp) 0]
set ftp1 [[set tcp1] attach-app FTP]
$ns at 0.8 "[set ftp1] start"
}
if {$opt(flows) > 0} {
set tcp1 [$ns create-connection TCP/Sack1 $nodes(is) TCPSink/Sack1 $nodes(lp) 0]
set ftp1 [[set tcp1] attach-app FTP]
$tcp1 set window_ 100
$ns at 0.0 "[set ftp1] start"
```

```
$ns at 3.5 "[set ftp1] stop"
set tcp2 [$ns create-connection TCP/Sack1 $nodes(is) TCPSink/Sack1 $nodes(lp) 0]
set ftp2 [[set tcp2] attach-app FTP]
$tcp2 set window_ 3
$ns at 1.0 "[set ftp2] start"
$ns at 8.0 "[set ftp2] stop"
}
proc stop {} {
global nodes opt nf
set wrap $opt(wrap)
set sid [$nodes($opt(srcTrace)) id]
set did [$nodes($opt(dstTrace)) id]
if {$opt(srcTrace) == "is"} {
set a "-a out.tr"
} else {
set a "out.tr"
}
set GETRC "../../../bin/getrc"
set RAW2XG "../../../bin/raw2xg"
exec $GETRC -s $sid -d $did -f 0 out.tr | \
$RAW2XG -s 0.01 -m $wrap -r >plot.xgr
exec $GETRC -s $did -d $sid -f 0 out.tr | \
$RAW2XG -a -s 0.01 -m $wrap >>plot.xgr
exec $GETRC -s $sid -d $did -f 1 out.tr | \
$RAW2XG -s 0.01 -m $wrap -r >>plot.xgr
exec $GETRC -s $did -d $sid -f 1 out.tr | \
$RAW2XG -s 0.01 -m $wrap -a >>plot.xgr
exec ./xg2gp.awk plot.xgr
if {!$opt(quiet)} {
exec xgraph -bb -tk -nl -m -x time -y packets plot.xgr&
}
exit 0
}
$ns at $opt(stop) "stop"
$ns run
```





OUTPUT :

EXPERIMENT 6

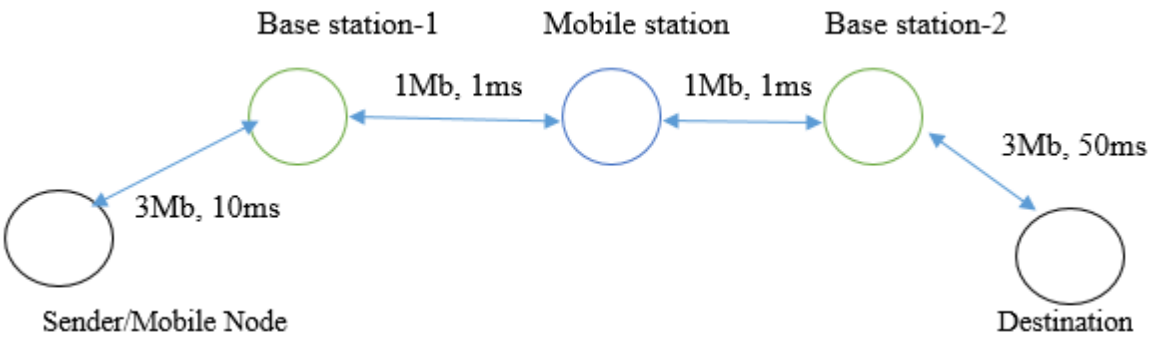
Implement and study the performance of CDMA on NS2/NS3 (Using stack called Call net) or equivalent environment.

3G networks developed as a replacement for second generation (2G) GSM standard network with full duplex voice telephony. CDMA is used as the access method in many mobile phone standards. IS-95, also called cdma One, and its 3G evolution CDMA2000, are often simply referred to as CDMA, but UMTS(The Universal Mobile Telecommunications System is a third generation mobile cellular system for networks based on the GSM standard.), the 3G standard used by GSM carriers, also uses wideband CDMA. Long-Term Evolution (LTE) is a standard for high-speed wireless communication which uses CDMA network technology.

3G technology generally refers to the standard of accessibility and speed of mobile devices. The standards of the technology were set by the International Telecommunication Union (ITU). This technology enables use of various services like GPS (Global Positioning System), mobile television and video conferencing. It not only enables them to be used worldwide, but also provides with better bandwidth and increased speed. The main aim of this technology is to allow much better coverage and growth with minimum investment.

CDMA can be implemented on all the versions of NS2 (Since year 2004: ns-2.27, and later versions of NS2)

Design:



Program:

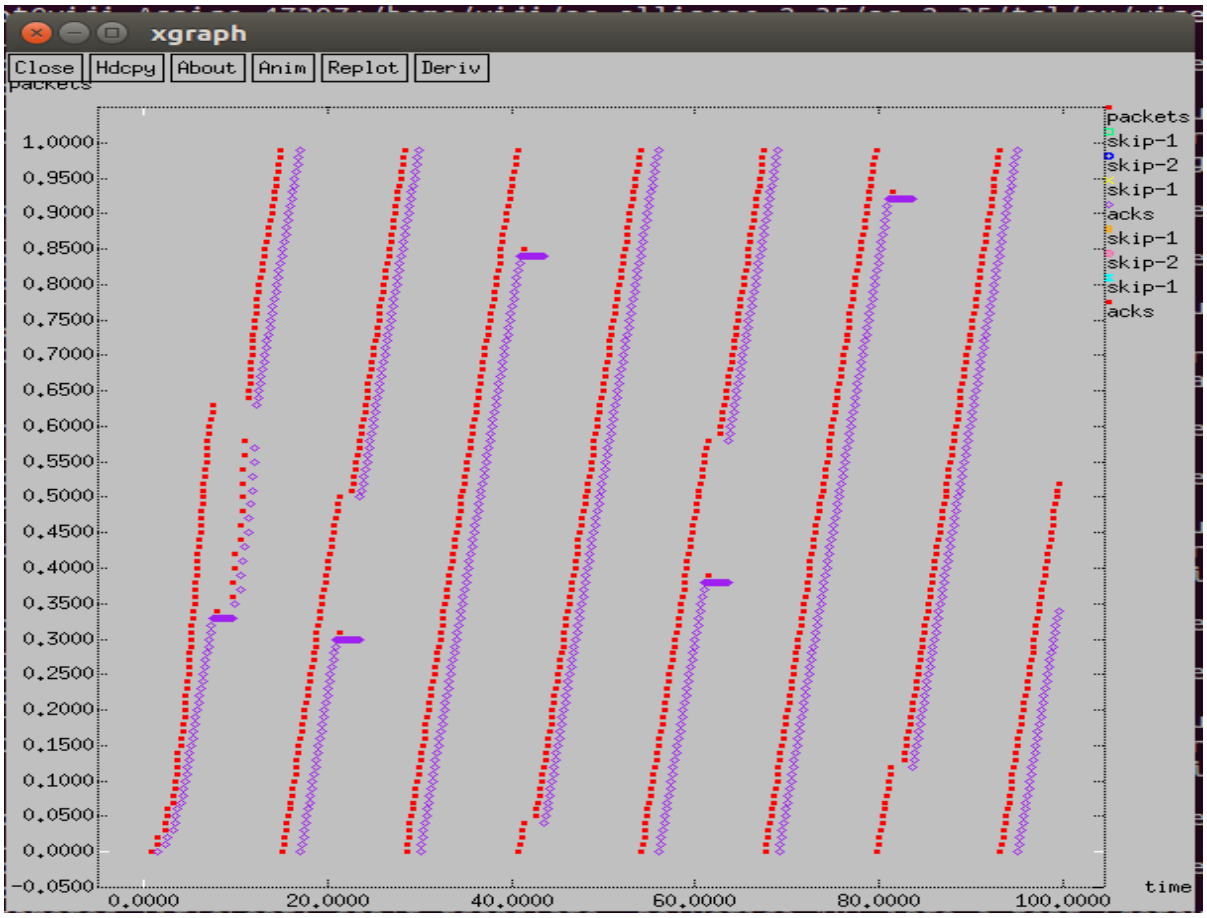
Source Code:

```
# General Parameters
set opt(title) zero ;
set opt(stop) 100 ;# Stop time.
set opt(ecn) 0 ;
# Topology
set opt(type) umts ;#type of link:
set opt(secondDelay) 55 ;# average delay of access links in ms
# AQM parameters
set opt(minth) 30 ;
set opt(maxth) 0 ;
set opt(adaptive) 1 ;# 1 for Adaptive RED, 0 for plain RED
# Traffic generation.
set opt(flows) 0 ;# number of long-lived TCP flows
set opt(window) 30 ;# window for long-lived traffic
set opt(web) 2 ;# number of web sessions
# Plotting statistics.
set opt(quiet) 0 ;# popup anything
set opt(wrap) 100 ;# wrap plots
set opt(srcTrace) is ;# where to plot traffic
set opt(dstTrace) bs2 ;# where to plot traffic
```

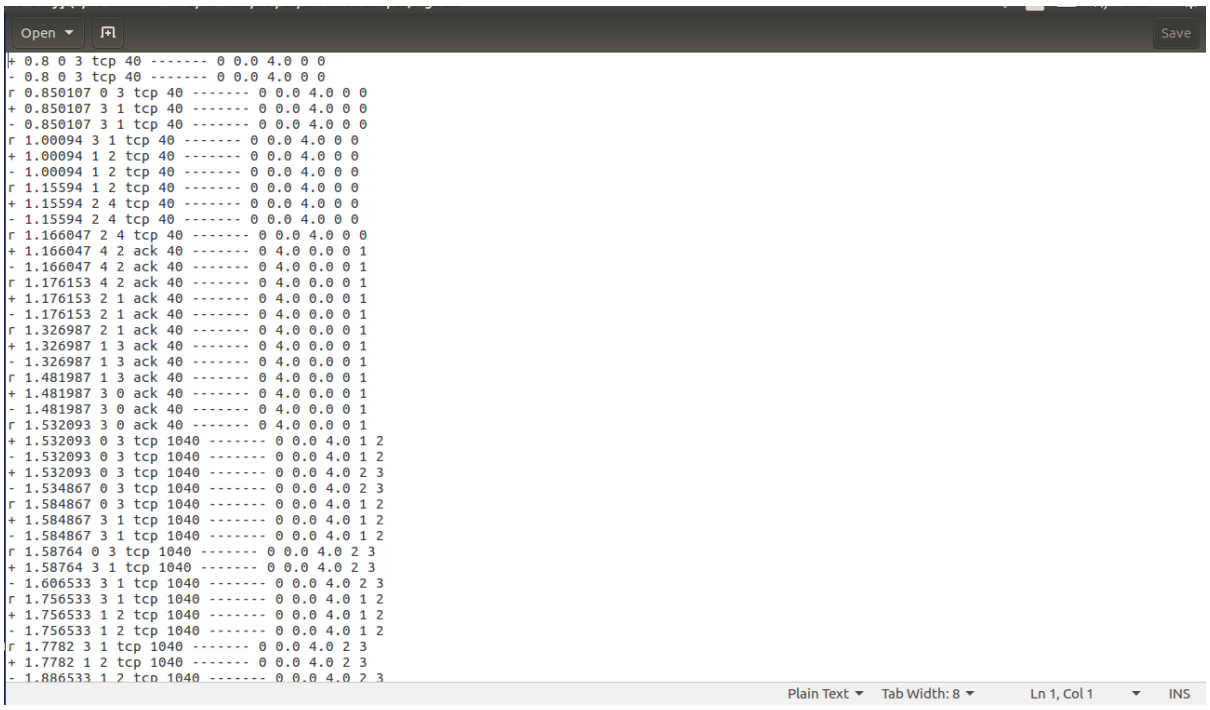
```
set opt(umtsbuf) 10 ; # buffer size for umts
#default downlink bandwidth in bps
set bwDL(umts) 384000
#default uplink bandwidth in bps
set bwUL(umts) 64000
#default downlink propagation delay in seconds
set propDL(umts) .150
#default uplink propagation delay in seconds
set propUL(umts) .150
#default buffer size in packets
set buf(umts) 20
set ns [new Simulator]
set tf [open out.tr w]
$ns trace-all $tf
set nodes(is) [$ns node]
set nodes(ms) [$ns node]
set nodes(bs1) [$ns node]
set nodes(bs2) [$ns node]
set nodes(lp) [$ns node]
proc cell_topo {} {
global ns nodes
$ns duplex-link $nodes(lp) $nodes(bs1) 3Mbps 10ms DropTail
$ns duplex-link $nodes(bs1) $nodes(ms) 1 1 RED
$ns duplex-link $nodes(ms) $nodes(bs2) 1 1 RED
$ns duplex-link $nodes(bs2) $nodes(is) 3Mbps 50ms DropTail
puts "Cell Topology"
}


proc set_link_params {t} {
global ns nodes bwULbwDLpropULpropDLbuf
$ns bandwidth $nodes(bs1) $nodes(ms) $bwDL($t) simplex
$ns bandwidth $nodes(ms) $nodes(bs1) $bwUL($t) simplex
$ns delay $nodes(bs1) $nodes(ms) $propDL($t) simplex
$ns delay $nodes(ms) $nodes(bs1) $propDL($t) simplex
$ns queue-limit $nodes(bs1) $nodes(ms) $buf($t)
$ns queue-limit $nodes(ms) $nodes(bs1) $buf($t)
$ns bandwidth $nodes(bs2) $nodes(ms) $bwDL($t) simplex
$ns bandwidth $nodes(ms) $nodes(bs2) $bwUL($t) simplex
$ns delay $nodes(bs2) $nodes(ms) $propDL($t) simplex
$ns delay $nodes(ms) $nodes(bs2) $propDL($t) simplex
$ns queue-limit $nodes(bs2) $nodes(ms) $buf($t)
$ns queue-limit $nodes(ms) $nodes(bs2) $buf($t)
}
# RED and TCP parameters
Queue/RED set summarystats_ true
Queue/DropTail set summarystats_ true
Queue/RED set adaptive_ $opt(adaptive)
Queue/RED set q_weight_ 0.0
Queue/RED set thresh_ $opt(minth)
Queue/RED set maxthresh_ $opt(maxth)
Queue/DropTail set shrink_drops_ true
Agent/TCP set ecn_ $opt(ecn)
Agent/TCP set window_ $opt(window)
DelayLink set avoidReordering_ true
```

```
source web.tcl
#Create topology
switch $opt(type) {
umts {cell_topo}
}
set_link_params $opt(type)
$ns insert-delayer $nodes(ms) $nodes(bs1) [new Delayer]
$ns insert-delayer $nodes(bs1) $nodes(ms) [new Delayer]
$ns insert-delayer $nodes(ms) $nodes(bs2) [new Delayer]
$ns insert-delayer $nodes(bs2) $nodes(ms) [new Delayer]
# Set up forward TCP connection
if {$opt(flows) == 0} {
set tcp1 [$ns create-connection TCP/Sack1 $nodes(is) TCPSink/Sack1 $nodes(lp) 0]
set ftp1 [[set tcp1] attach-app FTP]
$ns at 0.8 "[set ftp1] start"
}
if {$opt(flows) > 0} {
set tcp1 [$ns create-connection TCP/Sack1 $nodes(is) TCPSink/Sack1 $nodes(lp) 0]
set ftp1 [[set tcp1] attach-app FTP]
$tcp1 set window_ 100
$ns at 0.0 "[set ftp1] start"
$ns at 3.5 "[set ftp1] stop"
 set tcp2 [$ns create-connection TCP/Sack1 $nodes(is) TCPSink/Sack1 $nodes(lp) 0]
 set ftp2 [[set tcp2] attach-app FTP]
 $tcp2 set window_ 3
 $ns at 1.0 "[set ftp2] start"
 $ns at 8.0 "[set ftp2] stop"
 }
 proc stop {} {
 global nodes opt nf
 set wrap $opt(wrap)
 set sid [$nodes($opt(srcTrace)) id]
 set did [$nodes($opt(dstTrace)) id]
 if {$opt(srcTrace) == "is"} {
 set a "-a out.tr"
 } else {
 set a "out.tr"
 }
 set GETRC "../../../bin/getrc"
 set RAW2XG "../../../bin/raw2xg"
 exec $GETRC -s $sid -d $did -f 0 out.tr | \
 $RAW2XG -s 0.01 -m $wrap -r >plot.xgr
 exec $GETRC -s $did -d $sid -f 0 out.tr | \
 $RAW2XG -a -s 0.01 -m $wrap >>plot.xgr
 exec $GETRC -s $sid -d $did -f 1 out.tr | \
 $RAW2XG -s 0.01 -m $wrap -r >>plot.xgr
 exec $GETRC -s $did -d $sid -f 1 out.tr | \
 $RAW2XG -s 0.01 -m $wrap -a >>plot.xgr
 exec ./xg2gp.awk plot.xgr
 if {!$opt(quiet)} {
 exec xgraph -bb -tk -nl -m -x time -y packets plot.xgr&
 }
 exit 0
 }
 $ns at $opt(stop) "stop"
 $ns run
```

OUTPUT:



CDMA Trace File



Part B

EXPERIMENT 7

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

Theory

The cyclic redundancy check, or CRC, is a technique for detecting errors in digital data, but not for making corrections when errors are detected. It is used primarily in data transmission.

In the CRC method, a certain number of check bits, often called a checksum, are appended to the message being transmitted. The receiver can determine whether or not the check bits agree with the data, to ascertain with a certain degree of probability whether or not an error occurred in transmission.

It does error checking via polynomial division. In general, a bit string

$b_{n-1}\ b_{n-2}\ b_{n-3}\ \ldots b_2\ b_1\ b_0$

As

$b_{n-1}X^{n-1} + b_{n-2}X^{n-2} + b_{n-3}X^{n-3} + \ldots b_2X^2 + b_1X^1 + b_0$

Ex: -

1000100000100001

As

$X^{16} + X^{12} + X^5 + 1$

All computations are done in modulo 2

Algorithm:-

1. Given a bit string, append 0^S to the end of it (the number of 0^S is the same as the degree of the generator polynomial) let B(x) be the polynomial corresponding to B.
2. Divide B(x) by some agreed on polynomial G(x) (generator polynomial) and determine the remainder R(x). This division is to be done using Modulo 2 Division.
3. Define T(x) = B(x) –R(x)

   (T(x)/G(x) => remainder 0)
4. Transmit T, the bit string corresponding to T(x).
5. Let T' represent the bit stream the receiver gets and T'(x) the associated polynomial. The receiver divides T1(x) by G(x). If there is a 0 remainder, the receiver concludes T = T' and no error occurred otherwise, the receiver concludes an error occurred and requires a retransmission.

Figure 10.6: ***Division in CRC encoder***

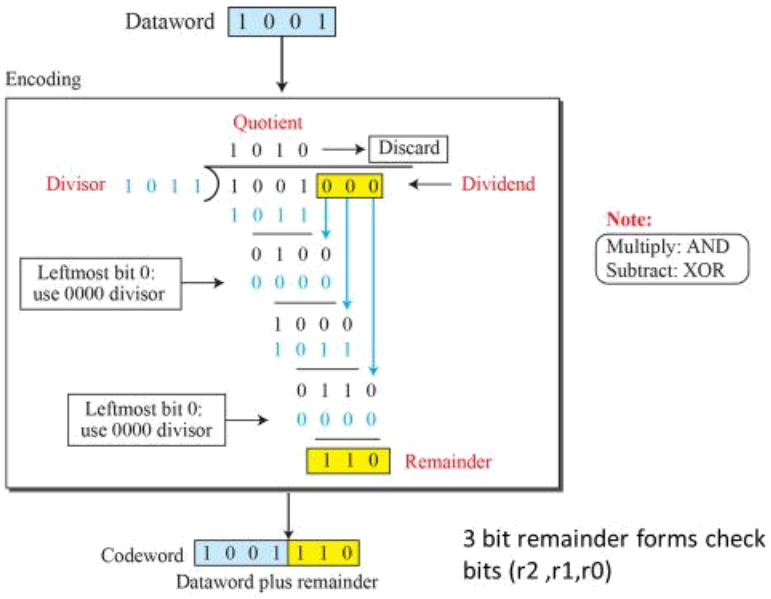
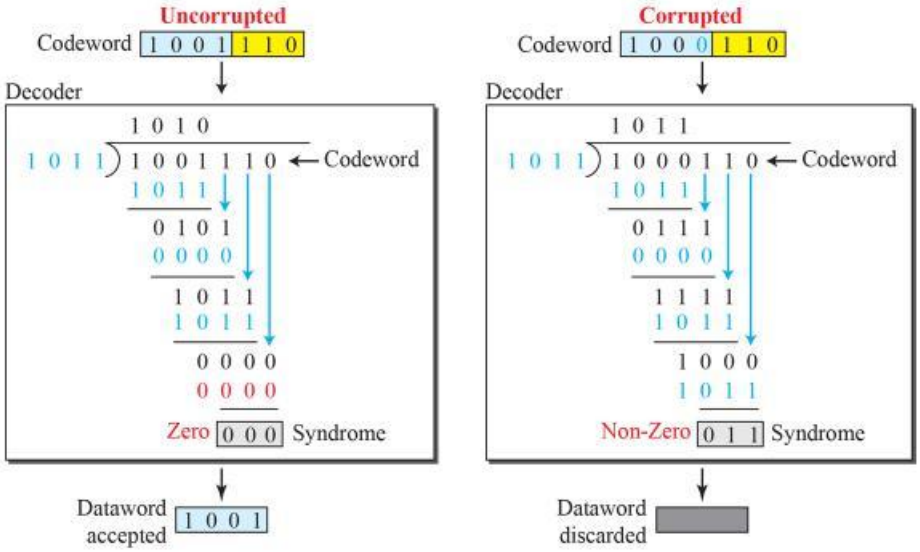


Figure 10.7: ***Division in the CRC decoder for two cases***



Program:

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

OUTPUT :

```
run:
Enter number of data bits :
7
Enter data bits :
1
0
1
1
0
0
1
Enter number of bits in divisor :
3
Enter Divisor bits :
1
0
1
Dividend (after appending 0's) are :
101100100

CRC code :
101100111
Enter CRC code of 9 bits :
1
0
1
1
0
0
1
0
1
crc bits are : 101100101
Error
THANK YOU.... :)
BUILD SUCCESSFUL (total time:
1 minute 34 seconds)
```

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

/*--------------------ERROR DETECTION----------------------*/
    System.out.println();
    System.out.println("Enter CRC code of "+tot_length+" bits : ");
    for(int i=0; i<crc.length; i++)
       crc[i]=Integer.parseInt(br.readLine());
     System.out.print("crc bits are : ");
    for(int i=0; i<crc.length; i++)
       System.out.print(crc[i]);
    System.out.println();
   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;
}}
```

EXPERIMENT 8

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

Theory

Routing algorithm is a part of network layer software which is responsible for deciding which output line an incoming packet should be transmitted on. If the subnet uses datagram internally, this decision must be made for every arriving data packet since the best route may have changed since last time. If the subnet uses virtual circuits internally, routing decisions are made only when a new established route is being set up. The latter case is sometimes called session routing, because a route remains in force for an entire user session (e.g., login session at a terminal or a file).

Routing algorithms can be grouped into two major classes: adaptive and nonadaptive. Nonadaptive algorithms do not base their routing decisions on measurement or estimates of current traffic and topology. Instead, the choice of route to use to get from I to J (for all I and J) is compute in advance, offline, and downloaded to the routers when the network ids booted. This procedure is sometime called static routing.

Adaptive algorithms, in contrast, change their routing decisions to reflect changes in the topology, and usually the traffic as well. Adaptive algorithms differ in where they get information (e.g., locally, from adjacent routers, or from all routers), when they change the routes (e.g., every ■*T* sec, when the load changes, or when the topology changes), and what metric is used for optimization (e.g., distance, number of hops, or estimated transit time).

Two algorithms in particular, distance vector routing and link state routing are the most popular. Distance vector routing algorithms operate by having each router maintain a table (i.e., vector) giving the best known distance to each destination and which line to get there. These tables are updated by exchanging information with the neighbors.

The distance vector routing algorithm is sometimes called by other names, including the distributed Bellman-Ford routing algorithm and the Ford-Fulkerson algorithm, after the researchers who developed it (Bellman, 1957; and Ford and Fulkerson, 1962). It was the original ARPANET routing algorithm and was also used in the Internet under the RIP and in early versions of DEC net and Novell's IPX. AppleTalk and Cisco routers use improved distance vector protocols.

In distance vector routing, each router maintains a routing table indexed by, and containing one entry for, each router in subnet. This entry contains two parts: the preferred out going line to use for that destination, and an estimate of the time or distance to that destination. The metric used might be number of hops, time delay in milliseconds, total number of packets queued along the path, or something similar.

The router is assumed to know the "distance" to each of its neighbor. If the metric is hops, the distance is just one hop. If the metric is queue length, the router simply examines each queue. If the metric is delay, the router can measure it directly with special ECHO packets hat the receiver just time stamps and sends back as fast as possible.

The Count to Infinity Problem.

Distance vector routing algorithm reacts rapidly to good news, but leisurely to bad news. Consider a router whose best route to destination X is large. If on the next exchange neighbor. A suddenly reports ashort delay to *X*, the router just switches over to using the line to *A* to send traffic to *X*. In one vector exchange, the good news is processed.

To see how fast good news propagates, consider the five node (linear) subnet of following figure, where the delay

metric is the number of hops. Suppose A is down initially and all the other routers know this. In other words, they have all recorded the delay to A as infinity.

A	B	C	D	E	A	B	C	D	E	
∞	∞		∞	∞	Initially	1	2	3	4	Initially
1	∞		∞	∞	After 1 exchange	3	2	3	4	After 1 exchange
1	2		∞	∞	After 2 exchange	3	3	3	4	After 2 exchange
1	2		3	∞	After 3 exchange	5	3	5	4	After 3 exchange
1	2		3	4	After 4 exchange	5	6	5	6	After 4 exchange
						7	6	7	6	After 5 exchange
						7	8	7	8	After 6 exchange
							:			
						∞	∞	∞	∞	

Many ad hoc solutions to the count to infinity problem have been proposed in the literature, each one more complicated and less useful than the one before it. The split horizon algorithm works the same way as distance vector routing, except that the distance to X is not reported on line that packets for X are sent on (actually, it is reported as infinity). In the initial state of right figure, for example, C tells D the truth about distance to A but C tells B that its distance to A is infinite. Similarly, D tells the truth to E but lies to C.

Each node x begins with $D_x(y)$, an estimate of the cost of the least-cost path from itself to node y, for all nodes in N. Let $D_x = [D_x(y): y \text{ in } N]$ be node x's distance vector, which is the vector of cost estimates from x to all other nodes, y, in N. With the DV algorithm, each node x maintains the following routing information:

For each neighbor v, the cost c(x,v) from x to directly attached neighbor, v

Node x's distance vector, that is, $D_x = [D_x(y): y \text{ in } N]$, containing x's estimate of its cost to all destinations, y, in N

The distance vectors of each of its neighbors, that is, $D_v = [D_v(y): y \text{ in } N]$ for each neighbor v of x

At each node, *x*:

```
1  Initialization:
2     for all destinations y in N:
3        Dx(y) = c(x,y)   /* if y is not a neighbor then c(x,y) = ∞ */
4     for each neighbor w
5        Dw(y) = ? for all destinations y in N
6     for each neighbor w
7        send distance vector Dx = [Dx(y): y in N] to w
8
9  loop
10    wait (until I see a link cost change to some neighbor w or
11          until I receive a distance vector from some neighbor w)
12
13    for each y in N:
14       Dx(y) = minv{c(x,v) + Dv(y)}
15
16    if Dx(y) changed for any destination y
17       send distance vector Dx = [Dx(y): y in N] to all neighbors
18
19 forever
```

Program:

```
import java.util.Scanner;

public class bellmanford
{
   private int           distances[];
   private int           numberofvertices;
   public static final int MAX_VALUE = 999;

   public bellmanford(int numberofvertices)
   {
this.numberofvertices = numberofvertices;
      distances = new int[numberofvertices + 1];
   }

   public void BellmanFordEvaluation(int source, int destination,
         int adjacencymatrix[][])
   {
      for (int node = 1; node <= numberofvertices; node++)
      {
         distances[node] = MAX_VALUE;
      }
      distances[source] = 0;
      for (int node = 1; node <= numberofvertices - 1; node++)
      {
         for (int sourcenode = 1; sourcenode<= numberofvertices; sourcenode++)
         {
            for (int destinationnode = 1; destinationnode<= numberofvertices; destinationnode++)
            {
               if (adjacencymatrix[sourcenode][destinationnode] != MAX_VALUE)
               {
                  if (distances[destinationnode] > distances[sourcenode]
                        + adjacencymatrix[sourcenode][destinationnode])
                     distances[destinationnode] = distances[sourcenode]
                           + adjacencymatrix[sourcenode][destinationnode];
               }
            }
         }
      }
      for (int sourcenode = 1; sourcenode<= numberofvertices; sourcenode++)
      {
         for (int destinationnode = 1; destinationnode<= numberofvertices; destinationnode++)
         {
            if (adjacencymatrix[sourcenode][destinationnode] != MAX_VALUE)
            {
               if (distances[destinationnode] > distances[sourcenode]
                     + adjacencymatrix[sourcenode][destinationnode])
System.out
```

```
.println("The Graph contains negative egde cycle");
                }
            }
        }
        for (int vertex = 1; vertex <= numberofvertices; vertex++)
        {
            if (vertex == destination)
System.out.println("distance of source  " + source + " to "
                    + vertex + " is " + distances[vertex]);
        }
    }

    public static void main(String... arg)
    {
        int numberofvertices = 0;
        int source, destination;
        Scanner scanner = new Scanner(System.in);
System.out.println("Enter the number of vertices");
numberofvertices = scanner.nextInt();
        int adjacencymatrix[][] = new int[numberofvertices + 1][numberofvertices + 1];
System.out.println("Enter the adjacency matrix");
        for (int sourcenode = 1; sourcenode<= numberofvertices; sourcenode++)
        {
            for (int destinationnode = 1; destinationnode<= numberofvertices; destinationnode++)
            {
adjacencymatrix[sourcenode][destinationnode] = scanner
.nextInt();
                if (sourcenode == destinationnode)
                {
adjacencymatrix[sourcenode][destinationnode] = 0;
                    continue;
                }
                if (adjacencymatrix[sourcenode][destinationnode] == 0)
                {
adjacencymatrix[sourcenode][destinationnode] = MAX_VALUE;
                }
            }
        }
System.out.println("Enter the source vertex");
        source = scanner.nextInt();
System.out.println("Enter the destination vertex: ");
        destination = scanner.nextInt();
bellmanfordbellmanford = new bellmanford(numberofvertices);
bellmanford.BellmanFordEvaluation(source, destination, adjacencymatrix);
scanner.close();
    }
}
```

OUTPUT:

```
run:
Enter the number of vertices
6
Enter the adjacency matrix
0 4 0 0 -1 0
0 0 -1 0 -2 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 -5 0 3
0 0 0 0 0 0
Enter the source vertex
1
Enter the destination vertex:
4
Distance of source 1 to 4 is -6
BUILD SUCCESSFUL (total time:
53 seconds)
```

EXPERIMENT 9

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.

TCP Socket

TCP is a connection-oriented protocol. This means that before the client and server can start to send data to each other, they first need to handshake and establish a TCP connection. One end of the TCP connection is attached to the client socket and the other end is attached to a server socket. When creating the TCP connection, we associate with it the client socket address (IPaddress and port number) and the server socket address (IPaddress and port number). With the TCP connection established, when one side wants to send data to the other side, it just drops the data into the TCP connection via its socket.

With the server process running, the client process can initiate a TCP connection to the server. This is done in the client program by creating a TCP socket. When the client creates its TCP socket, it specifies the address of the welcoming socket in the server, namely, the IP address of the server host and the port number of the socket. After creating its socket, the client initiates a three-way handshake and establishes a TCP connection with the server.

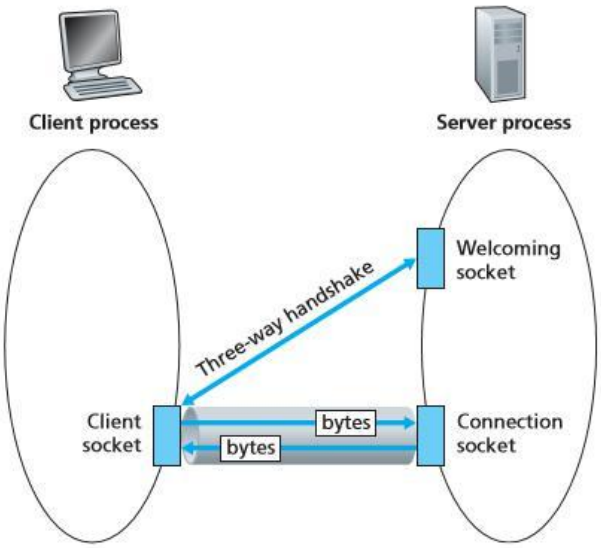
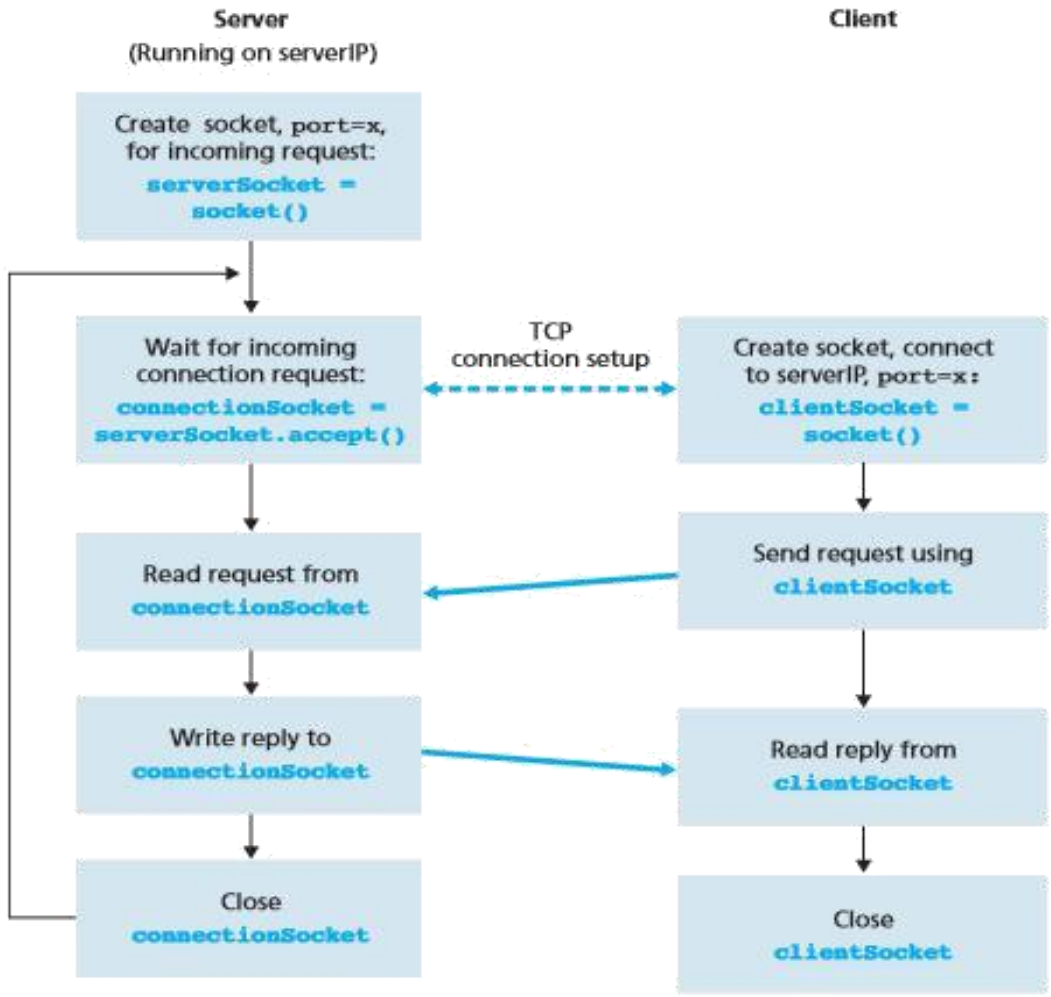


Figure 2.29 ♦ The TCPServer process has two sockets



Example: **Figure 2.30** ♦ The client-server application using TCP

Here is the code for the client side of the application:

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print 'From Server:', modifiedSentence
clientSocket.close()
```

Server Program

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while 1:
   connectionSocket, addr = serverSocket.accept()
   sentence = connectionSocket.recv(1024)
   capitalizedSentence = sentence.upper()
   connectionSocket.send(capitalizedSentence)
   connectionSocket.close()
```

Program: **Server1.java**

```
import java.io.*;
import java.net.*;
public class server1 {
 public static void main(String argv[]) throws Exception {
  String clientSentence;
  String capitalizedSentence;
ServerSocketwelcomeSocket = new ServerSocket(6789);
  while (true) {
   Socket connectionSocket = welcomeSocket.accept();
BufferedReaderinFromClient = new BufferedReader(new InputStreamReader
(connectionSocket.getInputStream()));
DataOutputStreamoutToClient = new DataOutputStream
(connectionSocket.getOutputStream());
clientSentence = inFromClient.readLine();
System.out.println("Received: " + clientSentence);
capitalizedSentence = clientSentence.toUpperCase() + '\n';
outToClient.writeBytes(capitalizedSentence);
  }
 }
}
```

Client1.java

```
import java.io.*;
import java.net.*;
public class client1 {
 public static void main(String argv[]) throws Exception {
  String sentence;
  String modifiedSentence;
BufferedReaderinFromUser = new BufferedReader(new
InputStreamReader(System.in));
  Socket clientSocket = new Socket("localhost", 6789);
DataOutputStreamoutToServer = new
DataOutputStream(clientSocket.getOutputStream());
BufferedReaderinFromServer = new BufferedReader(new InputStreamReader
(clientSocket.getInputStream()));
  sentence = inFromUser.readLine();
outToServer.writeBytes(sentence + '\n');
modifiedSentence = inFromServer.readLine();
System.out.println("FROM SERVER: " + modifiedSentence);
clientSocket.close();
 }}
```

OUTPUT: run both files Server1.java &Client1.java

Client1 side

```
run:
hi, good Morning, Are you fine?
FROM SERVER: HI, GOOD MORNING, ARE YOU FINE?
BUILD SUCCESSFUL (total time: 43 seconds)
```

Server1 side

```
run:
Received: hi, good Morning, are you fine?
```

EXPERIMENT 10

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

Using User Datagram Protocol, Applications can send data/message to the other hosts without prior communications or channel or path. This means even if the destination host is not available, application can send data. There is no guarantee that the data is received in the other side. Hence it's not a reliable service.

UDP is appropriate in places where delivery of data doesn't matters during data transition.

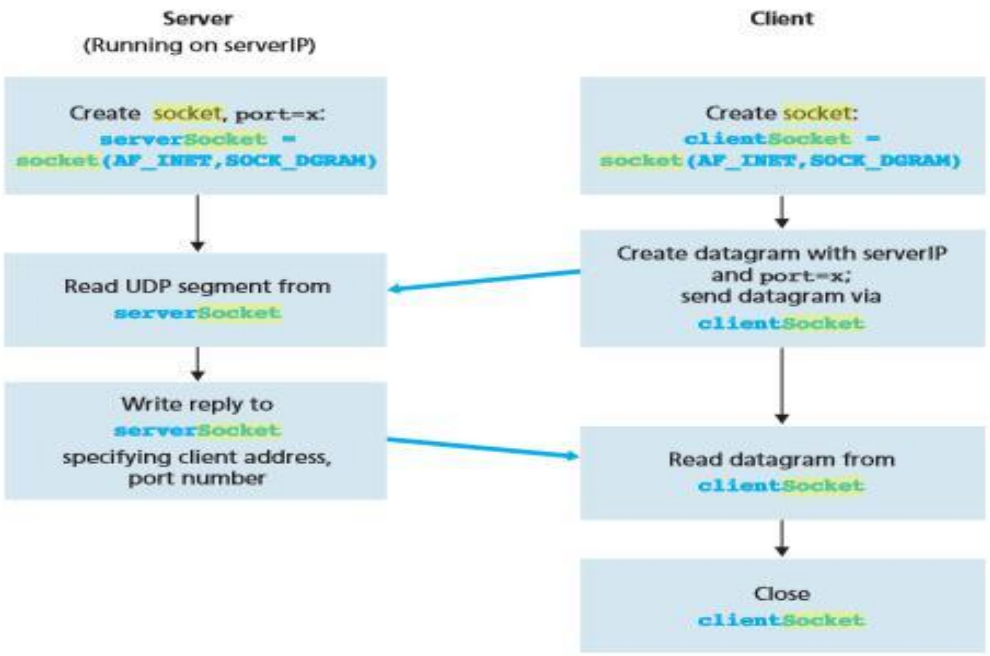


Figure 2.28 ♦ The client-server application using UDP

Server.java

Program:

```
import java.io.*;
import java.net.*;
public class server
{
    public static void main(String args[]) throws Exception
    {
DatagramSocketserverSocket = new DatagramSocket(9876);
byte[] receiveData = new byte[1024];
byte[] sendData = new byte[1024];
          while(true)
            {
```

```
DatagramPacketreceivePacket = new DatagramPacket(receiveData, receiveData.length);
serverSocket.receive(receivePacket);
            String sentence = new String( receivePacket.getData());
System.out.println("RECEIVED: " + sentence);
InetAddressIPAddress = receivePacket.getAddress();
            int port = receivePacket.getPort();
            String capitalizedSentence = sentence.toUpperCase();
sendData = capitalizedSentence.getBytes();
DatagramPacketsendPacket =
            new DatagramPacket(sendData, sendData.length, IPAddress, port);
serverSocket.send(sendPacket);
          }
    }
}
```

Client.java

```
import java.io.*;
import java.net.*;
public class client
{
  public static void main(String args[]) throws Exception
  {
BufferedReaderinFromUser =
      new BufferedReader(new InputStreamReader(System.in));
DatagramSocketclientSocket = new DatagramSocket();
InetAddressIPAddress = InetAddress.getByName("localhost");
byte[] sendData = new byte[1024];
byte[] receiveData = new byte[1024];
    String sentence = inFromUser.readLine();
sendData = sentence.getBytes();
DatagramPacketsendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, 9876);
clientSocket.send(sendPacket);
DatagramPacketreceivePacket = new DatagramPacket(receiveData, receiveData.length);
clientSocket.receive(receivePacket);
    String modifiedSentence = new String(receivePacket.getData());
System.out.println("FROM SERVER:" + modifiedSentence);
clientSocket.close();
  }
}
```

OUTPUT: **run both server.java & client.java files**

Client side

```
run:
hello,how r u
FROM SERVER:HELLO,HOW R U
```

Server side

```
RECEIVED: hello,how r u
```

EXPERIMENT-11

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

Theory

Cryptography has a long and colorful history. The message to be encrypted, known as the plaintext, are transformed by a function that is parameterized by a key. The output of the encryption process, known as the ciphertext, is then transmitted, often by messenger or radio. The enemy, or intruder, hears and accurately copies down the complete ciphertext. However, unlike the intended recipient, he does not know

the decryption key and so cannot decrypt the ciphertext easily. The art of breaking ciphers is called cryptanalysis the art of devising ciphers (cryptography) and breaking them (cryptanalysis) is collectively known as cryptology.

There are several ways of classifying cryptographic algorithms. They are generally categorized based on the number of keys that are employed for encryption and decryption, and further defined by their application and use. The three types of algorithms are as follows:

1. Secret Key Cryptography (SKC): Uses a single key for both encryption and decryption. It is also known as symmetric cryptography.
2. Public Key Cryptography (PKC): Uses one key for encryption and another for decryption. It is also known as asymmetric cryptography.
3. Hash Functions: Uses a mathematical transformation to irreversibly "encrypt" information

Public-key cryptography has been said to be the most significant new development in cryptography. Modern PKC was first described publicly by Stanford University professor Martin Hellman and graduate student Whitfield Diffie in 1976. Their paper described a two-key crypto system in which two parties could engage in a secure communication over a non-secure communications channel without having to share a secret key.

Generic PKC employs two keys that are mathematically related although knowledge of one key does not allow someone to easily determine the other key. One key is used to encrypt the plaintext and the other key is used to decrypt the ciphertext. The important point here is that it does not matter which key is applied first, but that both keys are required for the process to work. Because pair of keys is required, this approach is also called asymmetric cryptography.

In PKC, one of the keys is designated the public key and may be advertised as widely as the owner wants. The other key is designated the private key and is never revealed to another party. It is straight forward to send messages under this scheme.

The RSA algorithm is named after Ron Rivest, Adi Shamir and Len Adleman, who invented it in 1977. 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.

Algorithm

1. Generate two large random primes, P and Q, of approximately equal size.
2. Compute $N = P \times Q$
3. Compute $Z = (P-1) \times (Q-1)$.
4. Choose an integer E, $1 < E < Z$, such that GCD $(E, Z) = 1$
5. Compute the secret exponent D, $1 < D < Z$, such that $E \times D \equiv 1 \pmod{Z}$
6. The public key is (N, E) and the private key is (N, D).

Note: The values of *P*, *Q*, and *Z* should also be kept secret.
The message is encrypted using public key and decrypted using private key.

An example of RSA encryption

1. Select primes *P*=11, *Q*=3.
2. $N = P \times Q = 11 \times 3 = 33$ $Z = (P-1) \times (Q-1) = 10 \times 2 = 20$
3. Lets 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*, *Z*) = GCD(3, 20) = 1
4. Compute *D* such that $E \times D \equiv 1 \pmod{Z}$ compute $D = E^{\wedge -1} \bmod Z = 3^{\wedge -1} \bmod 20$
   find a value for *D* such that *Z* divides ((*E*x*D*)-1) find *D* such that 20 divides 3D-1.

   Simple testing (D = 1, 2, ...) gives D = 7
   Check: (*E* x *D*)-1 = 3.7 - 1 = 20, which is divisible by *Z*.
5. Public key = (*N*, *E*) = (33, 3)
   Private key = (*N*, *D*) = (33, 7).

Now say we want to encrypt the message m = 7,
Cipher code = $M^{\wedge E} \bmod N$
$= 7^{\wedge 3} \bmod 33$
$= 343 \bmod 33 = 13$.
Hence the ciphertext c = 13.

To check decryption we compute Message' = $C^{\wedge D} \bmod N = 13^{\wedge 7} \bmod 33 = 7$.

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

Program:

```
import java.math.BigInteger;
import java.security.*;
import java.security.spec.*;
import java.io.*;
import javax.crypto.Cipher;
public class rsa {
public static void main(String args[])
{
String srci="";
try{
BufferedReaderbr=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Please enter any string you want to encrypt");
srci=br.readLine();
}
catch(IOExceptionioe)
{
System.out.println(ioe.getMessage());
}
try{
KeyPairGeneratorkpg=KeyPairGenerator.getInstance("RSA");
kpg.initialize(512);//initialize key pairs to 512 bits ,you can also take 1024 or 2048 bits
KeyPairkp=kpg.genKeyPair();
PublicKeypubli=kp.getPublic();
Cipher cipher = Cipher.getInstance("RSA");
cipher.init(Cipher.ENCRYPT_MODE, publi);
byte[]src=srci.getBytes();//converting source data into byte array
byte[] cipherData = cipher.doFinal(src);//use this method to finally encrypt data
String srco=new String(cipherData);//converting byte array into string
System.out.println();
System.out.println("Encrypted data is:-"+srco);
PrivateKeyprivatei=kp.getPrivate();//Generating private key
Cipher cipheri=Cipher.getInstance("RSA");//Intializing 2nd instance of Cipher class
cipheri.init(Cipher.DECRYPT_MODE, privatei);//Setting to decrypt_mode
byte[] cipherDat = cipheri.doFinal(cipherData);//Finally decrypting data
String decryptdata=new String(cipherDat);
System.out.println("Decrypted data:-"+decryptdata);
}
catch(Exception e)
{
System.out.println(e.getMessage());
}}}
```

OUTPUT:-

```
Please enter any string you want to encrypt
hello
Encrypted data is:-i`>??a?:?ls
Decrypted data:-hello
BUILD SUCCESSFUL (total time: 12 seconds)
```

EXPERIMENT 12

Write a program for congestion control using leaky bucket algorithm.

Theory

The congesting control algorithms are basically divided into two groups: open loop and closed loop. Open loop solutions attempt to solve the problem by good design, in essence, to make sure it does not occur in the first place. Once the system is up and running, midcourse corrections are not made. Open loop algorithms are further divided into ones that act at source versus ones that act at the destination.

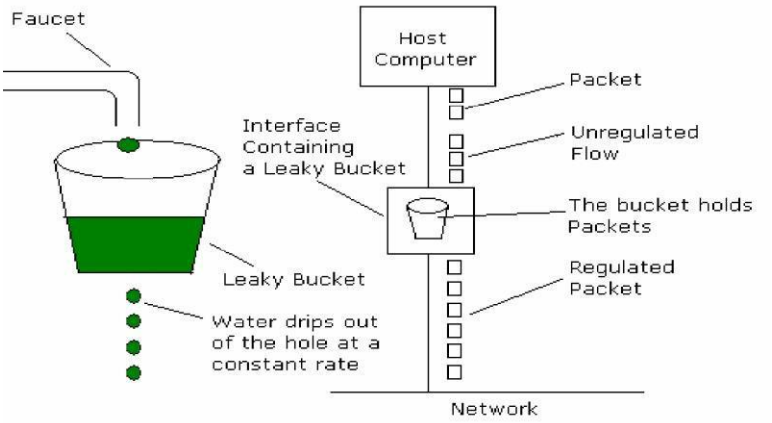
In contrast, closed loop solutions are based on the concept of a feedback loop if there is any congestion. Closed loop algorithms are also divided into two sub categories: explicit feedback and implicit feedback. In explicit feedback algorithms, packets are sent back from the point of congestion to warn the source. In implicit algorithm, the source deduces the existence of congestion by making local observation, such as the time needed for acknowledgment to come back.

The presence of congestion means that the load is (temporarily) greater than the resources (in part of the system) can handle. For subnets that use virtual circuits internally, these methods can be used at the network layer.

Another open loop method to help manage congestion is forcing the packet to be transmitted at a more predictable rate. This approach to congestion management is widely used in ATM networks and is called traffic shaping.

The other method is the leaky bucket algorithm. Each host is connected to the network by an interface containing a leaky bucket, that is, a finite internal queue. If a packet arrives at the queue when it is full, the packet is discarded. In other words, if one or more process are already queued, the new packet is unceremoniously discarded. This arrangement can be built into the hardware interface or simulate d by the host operating system. In fact it is nothing other than a single server queuing system with constant service time.

The host is allowed to put one packet per clock tick onto the network. This mechanism turns an uneven flow of packet from the user process inside the host into an even flow of packet onto the network, smoothing out bursts and greatly reducing the chances of congestion.



Program:

```
import java.util.*;
public class leakybucketalgorithm {
   public static void main(String[] args)
   {
      Scanner my = new Scanner(System.in);
      int no_groups,bucket_size;
System.out.print("\n Enter the bucket size : \t");
bucket_size = my.nextInt();
System.out.print("\n Enter the no of groups : \t");
no_groups = my.nextInt();
      int no_packets[] = new int[no_groups];
      int in_bw[] = new int[no_groups];
      int out_bw,reqd_bw=0,tot_packets=0;
for(int i=0;i<no_groups;i++)
      {
System.out.print("\n Enter the no of packets for group " + (i+1) + "\t");
no_packets[i] = my.nextInt();
System.out.print("\n Enter the input bandwidth for the group " + (i+1) + "\t");
in_bw[i] = my.nextInt();
         if((tot_packets+no_packets[i])<=bucket_size)
         {
tot_packets += no_packets[i];
         }
         else
         {
            do
            {
System.out.println(" Bucket Overflow ");
System.out.println(" Enter value less than " + (bucket_size-tot_packets));
no_packets[i] = my.nextInt();
}while((tot_packets+no_packets[i])>bucket_size);
tot_packets += no_packets[i];
         }
reqd_bw += (no_packets[i]*in_bw[i]);
      }
System.out.println("\nThe total required bandwidth is " + reqd_bw);
System.out.println("Enter the output bandwidth ");
out_bw = my.nextInt();
      int temp=reqd_bw;
      int rem_pkts = tot_packets;
      while((out_bw<=temp)&&(rem_pkts>0))
      {
System.out.println("Data Sent \n" + (--rem_pkts) + " packets remaining");
System.out.println("Remaining Bandwidth " + (temp -= out_bw));
         if((out_bw>temp)&&(rem_pkts>0))
System.out.println(rem_pkts + " packet(s) discarded due to insufficient bandwidth");
      }
   }
}
```

OUTPUT:-

Enter the bucket size : 10

Enter the no of groups: 2

Enter the no of packets for group 1 3

Enter the input bandwidth for the group 1 4

Enter the no of packets for group 2 4

Enter the input bandwidth for the group 2 3

The total required bandwidth is 24
Enter the output bandwidth

6
Data Sent
6 packets remaining
Remaining Bandwidth 18
Data Sent
5 packets remaining
Remaining Bandwidth 12
Data Sent
4 packets remaining
Remaining Bandwidth 6
Data Sent
3 packets remaining
Remaining Bandwidth 0
3 packet(s) discarded due to insufficient bandwidth
BUILD SUCCESSFUL (total time: 43 seconds)