BANGALORE INSTITUTE OF TECHNOLOGY

K.R. ROAD, V.V PURAM, BANGALORE – 560 004

DEPARTMENT OF
INFORMATION SCIENCE AND ENGINEERING

COMPUTER NETWORKS

SUBJECT CODE: BCS502

Outcome Based Education (OBE) and Choice Based Credit System (CBCS)

FOR V SEMESTER CSE/ISE AS PRESCRIBED BY VTU

Effective from the Academic year 2025-2026

Prepared By:
Dr. Jayasheela C. S**,**
Assistant Professor
Dept. of ISE, BIT

BANGALORE INSTITUTE OF TECHNOLOGY

VISION

- To Establish and Develop the Institute as a Center of Higher Learning ever abreast with the Expanding horizon of knowledge in the field of Engineering and Technology, with entrepreneurial thinking, leadership excellence for life-long success and solve societal problems.

MISSION

- Provide high quality education in the Engineering disciplines from the undergraduate through doctoral levels with creative academic and professional programs.

- Develop the Institute as a leader in Science, Engineering, Technology, Management and Research and apply knowledge for the benefit of society.

- Establish mutual beneficial partnership with Industry, Alumni, Local, State and Central Governments by Public Services Assistance and Collaborative Research.

- Inculcate personality development through sports, cultural and extracurricular activities and engage in the social, economic and professional challenges.

Bangalore Institute of Technology
K R Road, VV Puram, Bangalore- 560004
Department of Information Science and Engineering

VISION:

Empower every student to be innovative, creative and productive in the field of Information Technology by imparting quality technical education, **developing Professional Skills** and inculcating human values.

MISSION:

- To evolve continually as a centre of excellence in offering quality Information Technology **Education.**

- To nurture the students to meet the global competency in industry for **Employment.**

- To promote collaboration with industry and academia for constructive interaction to empower **Entrepreneurship.**

- To provide reliable, contemporary and integrated technology to support and facilitate **Life Long Learning.**

PROGRAM EDUCATIONAL OBJECTIVES

- Uplift the students through Information Technology **Education.**

- Provide exposure to emerging technologies and train them to **Employable** in Multi-disciplinary industries.

- Motivate them to become good professional Engineers and **Entrepreneur.**

- Inspire them to prepare for **Higher Learning and Research**.

PROGRAMME SPECIFIC OUTCOMES (PSOs)

- To provide our graduates **with Core Competence in Information Technology and Management.**

- To prepare our graduates with **relevant skills for Higher Education, Entrepreneurship, Professional career & Social values.**

PROGRAM OUTCOMES (POs)

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 a n d 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 t h e 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.

Bangalore Institute of Technology

K.R. Road, V.V. Pura, Bengaluru - 560004

Department of Information Science and Engineering

Prerequisites:

- Knowledge of computer networking concepts.
- Knowledge about JAVA programming Languages.
- Basic Concepts of Network Simulation tool

Course Objectives:

This course will enable students to:

- CLO 1. Fundamentals of data communication networks.
- CLO 2. Software and hardware interfaces
- CLO 3. Application of various physical components and protocols
- CLO 4. Communication challenges and remedies in the networks.

Course Outcomes:

At the end of the course the student will be able to:

- CO 1. Understand the basic needs of communication networks, types, models.
- CO 2. Apply error detection and correction methods, routing algorithms and congestion control algorithms.
- CO 3. Analyze data link layer and network layer issues, transport service and principles of network applications.
- CO 4. Implement and test point to point network, ESS, CRC code, Transmission of ping messages, Bellman-ford algorithms, Leaky bucket algorithms and Ethernet LAN.
- CO 5. Advance and latest topics in computer networks to be presented and documented (Content beyond the syllabus).

Mapping of COs-POs and COs-PSOs
Computer Networks Laboratory (BCS502)
Year of Study: 2024 -2025 (ODD)

CO to PO & PSO MAPPING

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
CO1	3	2											3	2
CO2	3	2											3	2
CO3	3	2											3	2
CO4	3	3	2		3								3	2
CO5	3	3			2				3	3		2	3	2

COMPUTER NETWORKS		**Semester**	V
Course Code:	BCS502	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	3:0:2:0	SEE Marks	50
Total Hours of Pedagogy	40 hours Theory + 8-10 Lab slots	Total Marks	100
Credits	04	Exam Hours	03
Examination nature (SEE)	Theory/practical		

List of Programs

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

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

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

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.

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

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

10. Develop a program for congestion control using a leaky bucket algorithm.

Practical Sessions need to be assessed by appropriate rubrics and viva-voce method. This will contribute to 25 marks
- Daily conduction with record – 15
- Test conduction with viva-voce - 10

RUBRIC SHEET

DAILY CONDUCTION (Max: 15 Marks)

Sl. No	Experiment Name	Write-Up & Implementation 3 Marks	Analysis & Execution 4 Marks	Results & Tabulation 3 Marks	Record 5 Marks	Total 15 Marks
1	Write a program for error detecting code using CRC-CCITT (16- bits).					
2	Develop a program to implement a sliding window protocol in the data link layer.					
3	Develop a program to find the shortest path between vertices using the Bellman-Ford and path vector routing algorithm.					
4	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.					
5	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.					
6	Develop a program on a datagram socket for client/server to display the messages on client side, typed at the server side.					
7	Develop a program for a simple RSA algorithm to encrypt and decrypt the data.					
8	Develop a program for congestion control using a leaky bucket algorithm					
9	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.					
10	Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.					

Test (T1+T2) Rubrics (Max: 10 Marks)

	Write-up & Implementation (10Marks)	Analysis & Execution (20Marks)	Results & Tabulation (10Marks)	Viva (10Marks)	Total 50 Marks
Test-1					
Test-2					
	T1+T2 (100 Marks scaled down 10 Marks)			10	
	Daily conduction + Test Marks (15 + 10)			25	

Computer Network Laboratory

Procedure for Program Execution

ENVIRONMENT: **ECLIPSE IDE TOOL**

I. CREATION OF PACKAGE (ONE TIME CREATION FORRECURRENT USAGE IN ALL LABS)

- GO TO FILE **->** NEW **->** PACKAGE
- Provide location for package creation
- Give a name to the package to be created **->** select finish

II. STEPS TO CREATE PROGRAM

- Go to package explorer (lhs)
- Select your package in the specified location
- Right click on the package name
- Select new **->** class **->** provide name for the main class **->** finish
- Your new default main class will be created.
- Edit this file to insert your new java code and save the file

III. STEPS FOR COMPILATION

- Syntax errors will show up on the left with an „x" icon
- By placing a mouse over an error icon, you can see a suggestion boxthat lists the ways you can fix the error

IV. STEPS FOR EXECUTION

- Click on the triangular icon to run your program

  or

- Select "run" from the main menu and then select "run" again from thedrop down menu

  or

- Ctrl+f11

Procedure for ns2 program execution

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

   [root@localhost ~]# gedit lab1.tcl/vi lab1.tcl

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

   [root@localhost ~]# gdit lab1.awk/vi lab1.awk

3) Run the simulation program

   [root@localhost~]# ns lab1.tcl

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

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

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

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

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

   [root@localhost~]# gedit lab1.tr/vi lab1.tr

SIMULATION USING NS-2

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., routingalgorithms, TCP, UDP) can be done using NS2.
- In general, NS2 provides users with a way of specifying such network protocols andsimulating their corresponding behaviors

Basic Architecture of NS2
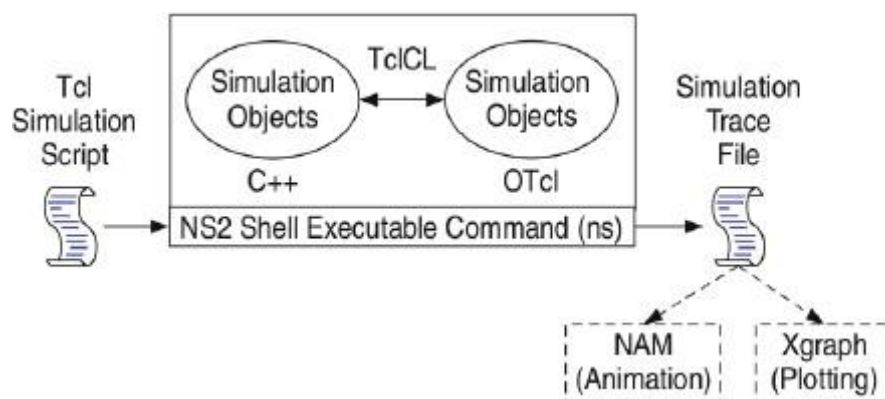
Tcl scripting

• Tcl is a general purpose scripting language. [Interpreter]

• Tcl runs on most of the platforms such as Unix, Windows, and Mac.

• The strength of Tcl is its simplicity.

• It is not necessary to declare a data type for variable prior to the usage.

Basics of TCL

Syntax: command   arg1   arg2   arg3

- **Hello World!**
  puts stdout{Hello,
  World!}Hello,
  World!

- **Variables**   Command Substitution

  set a 5   set len [string length foobar]

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

- **Simple Arithmetic**
  expr 7.2 / 4

- **Procedures**

  proc Diag {a b} {

  set c [expr sqrt($a * $a +

  $b * $b)]return $c }

  puts ―Diagonal of a 3, 4 right triangle

  is[Diag 3 4]Output: Diagonal of a 3, 4 right

  triangle is 5.0

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

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 installationof the class Simulator using the reserved word new.

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

#Open the trace file

```
set tracefile1 [open out.tr w]
$ns trace-all $tracefile1
```

#Open the NAM trace file

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

The above creates a trace file called ―out.tr and a nam visualization trace file called ―out.nam. Within the tcl script, these files are not called explicitly by their names, but instead by pointers that are declared above and called ―tracefile1 and ―namfile respectively. Remark that they begins with a # symbol. The second line open the file ―out.tr to be used for writing, declared with the letter ―w. The third line uses a simulator method called trace-all that have as parameterthe 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 **finish** is called without arguments. The word **global** is used to tell that we are using variables declared outside the procedure. The simulator method ―**flush-trace"** will dump the traces on the respective files. The tcl command ―**close"** closes the trace files defined before and **exec** executes the nam program for visualization. The command **exit** will ends the application and return the number 0 as status to the system. Zero is the default for a clean exit. Other values can be used to say that is exit because something fails.

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

> **$ns at 125.0 "finish"**

will be used to call ―**finish** at time 125sec.Indeed,the **at** method of the simulator allows us toschedule 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 ofpropagation delay and a capacity of 10Mb per sec for each direction.

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

In NS, an output queue of a node is implemented as a part of each link whose input is that node. The definition of the link then includes the way to handle overflow at that queue. In our case, if the buffer capacity of the output queue is exceeded then the last packet to arrive is dropped. Many alternative options exist, such as the RED (Random Early Discard)mechanism, the FQ (Fair Queuing), the DRR (Deficit Round Robin), the stochastic Fair Queuing (SFQ) and the CBQ (which including a priority and a round-robin scheduler).

In ns, an output queue of a node is implemented as a part of each link whose input is that node. We should also define the buffer capacity of the queue related to each link. An example would be:

```
#set Queue Size of link (n0-n2) to 20
$ns queue-limit $n0 $n2 20
```

Agents and Applications

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

FTP over TCP

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

There are number variants of the TCP protocol, such as Tahoe, Reno, NewReno, Vegas. Thetype 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. Thecommand.

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

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

#Setup a UDP connection

```
set udp [new Agent/UDP]

$ns attach-agent $n1 $udp

set null [new Agent/Null]

$ns attach-agent $n5 $null

$ns connect $udp $null

$udp set fid_2
```

#setup a CBR over UDP connection

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

$cbr attach-agent $udp

$cbr set packetsize_ 100

$cbr set rate_ 0.01Mb

$cbr set random_ false
```

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

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

TCP has many parameters with initial fixed defaults values that can be changed if mentioned explicitly. For example, the default TCP packet size has a size of 1000bytes.This can be changed to another value, say 552bytes, using the command **$tcp set packetSize_552**.

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

CBR over UDP

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

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

```
$cbr set interval_ 0.005
```

The packet size can be set to some value using

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

Scheduling Events

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

$ns at <time><event>

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

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

$ns at 0.1 "$cbr start"

$ns at 1.0 " $ftp start"

$ns at 124.0 "$ftp stop"

$ns at 124.5 "$cbr stop"

Structure of Trace Files

When tracing into an output ASCII file, the trace is organized in 12 fields as follows in fig shownbelow, 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.
- The second field gives the time at which the event occurs.
- Gives the input node of the link at which the event occurs.
- Gives the output node of the link at which the event occurs.
- Gives the packet type (eg:- CBR or TCP)
- Gives the packet size
- Some flags
- 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.
- This is the source address given in the form of ─node.port.

- This is the destination address, given in the same form.
- 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
- 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.lstVariables

Awk allows the user to use variables of their choice. You can now print a serial number, usingthe variable kount, and apply it those directors drawing a salary exceeding 6700:

$ awk –F"|" „$3 == "director" && $6 > 6700 {kount =kount+1
printf " %3f %20s %-12s %d\n", kount,$2,$3,$6 }" empn.lst

THE –f OPTION: STORING awk PROGRAMS IN A FILE

You should holds large awk programs in separate file and provide them with the awk extensionfor 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 having6 fields, and which have crept in due to faulty data entry:

$awk „BEGIN {FS = “|”}

NF! =6 {Print “Record No “, NR, “has”, “fields”}}“ empx.lst

Program 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.

Program:

#Create a simulator object

set ns [new Simulator]

set tf [open lab1.tr w]

$ns trace-all $tf

#Open the NAM trace file

set nf [open lab1.nam w]

$ns namtrace-all $nf

#Create four nodes

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

#Define different colors for data flows (for NAM)

$ns color 1 "red"

$ns color 2 "blue"

$n0 label "source/udp0"

$n1 label "source/udp1"

$n2 label "Router"

$n3 label "Destination/Null"

#Create links between the nodes

$ns duplex-link $n0 $n2 10Mb 300ms DropTail

$ns duplex-link $n1 $n2 10Mb 300ms DropTail

$ns duplex-link $n2 $n3 1Mb 300ms DropTail

#Set Queue Size of link

$ns set queue-limit $n0 $n2 10

$ns set queue-limit $n1 $n2 10

$ns set queue-limit $n2 $n3 5

set udp0 [new Agent/UDP]

$ns attach-agent $n0 $udp0

#Setup a CBR over UDP connection

set cbr0 [new Application/Traffic/CBR]

$cbr0 attach-agent $udp0

set null3 [new Agent/Null]

$ns attach-agent $n3 $null3

set udp1 [new Agent/UDP]

$ns attach-agent $n1 $udp1

set cbr1 [new Application/Traffic/CBR]

$cbr1 attach-agent $udp1

$udp0 set class_ 1

$udp1 set class_ 2

#Setup a UDP connection

$ns connect $udp0 $null3

$ns connect $udp1 $null3

$cbr1 set packetSize_ 500Mb

$cbr1 set interval_ 0.005

#Define a 'finish' procedure

```
proc finish {} {
                global ns nf tf
                $ns flush-trace
                exec nam lab1.nam &
                exec echo "The number of packet drop is:" &
                exec grep -c "^d" lab1.tr &
                close $tf
                close $nf
                exit 0
        }
```

 #Schedule events for the CBR agents

$ns at 0.1 "$cbr0 start"

$ns at 0.1 "$cbr1 start"

#Call the finish procedure after 10 seconds of simulation time

$ns at 10.0 "finish"

#Run the simulation

$ns run

Steps for execution

Open geditor and type program. Program name should have the extension " .tcl"

[root@localhost ~]# gedit lab1.tcl / vi lab1.tcl

- Run the simulation program

[root@localhost~]# ns lab1.tcl

  - Here **"ns"** indicates network simulator. We get the topology shown in the snapshot.
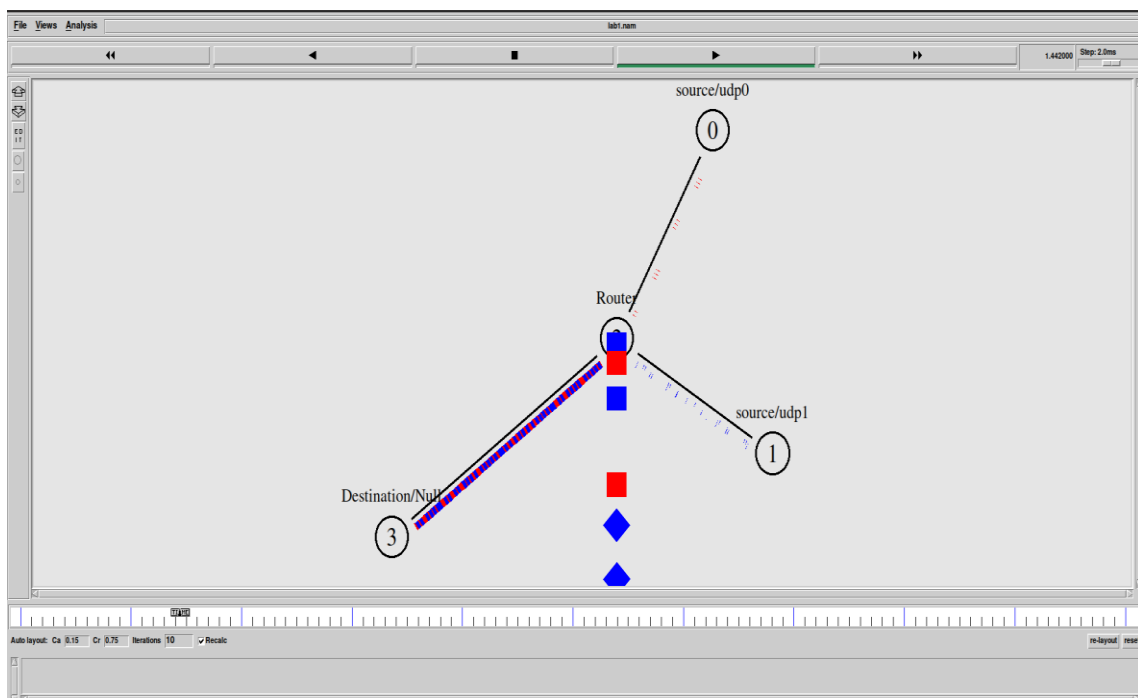 - Now press the play button in the simulation window and the simulation will begins.

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

[root@localhost~]# gedit lab1.tr / vi lab1.tr

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 ID

Topology:

Note:

1. Set the queue size fixed from n0 to n2 as 10, n1-n2 to 10 and from n2-n3as5. 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 find the number ofpackets dropped at the node2

Program 2

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

Program:

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

set nf [open lab2.nam w]
$ns namtrace-all $nf

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]

$n0 label "Ping0"
$n4 label "Ping4"
$n5 label "Ping5"
$n6 label "Ping6"
$n2 label "Router"

$ns color 1 "red"
$ns color 2 "green"
$ns duplex-link $n0 $n2 100Mb 300ms DropTail
$ns duplex-link $n1 $n2 1Mb 300ms DropTail
$ns duplex-link $n3 $n2 1Mb 300ms DropTail
$ns duplex-link $n5 $n2 100Mb 300ms DropTail
$ns duplex-link $n2 $n4 1Mb 300ms DropTail
$ns duplex-link $n2 $n6 1Mb 300ms DropTail
$ns queue-limit $n0 $n2 5
$ns queue-limit $n2 $n4 3
$ns queue-limit $n2 $n6 2
$ns queue-limit $n5 $n2 5
```

#The below code is used to connect between the ping agents to the node n0, n4 , n5 and n6.

set ping0 [new Agent/Ping]

$ns attach-agent $n0 $ping0

set ping4 [new Agent/Ping]

$ns attach-agent $n4 $ping4

set ping5 [new Agent/Ping]

$ns attach-agent $n5 $ping5

set ping6 [new Agent/Ping]

$ns attach-agent $n6 $ping6

$ping0 set packetSize_ 50000

$ping0 set interval_ 0.0001

$ping5 set packetSize_ 60000

$ping5 set interval_ 0.00001

$ping0 set class_ 1

$ping5 set class_ 2

$ns connect $ping0 $ping4

$ns connect $ping5 $ping6
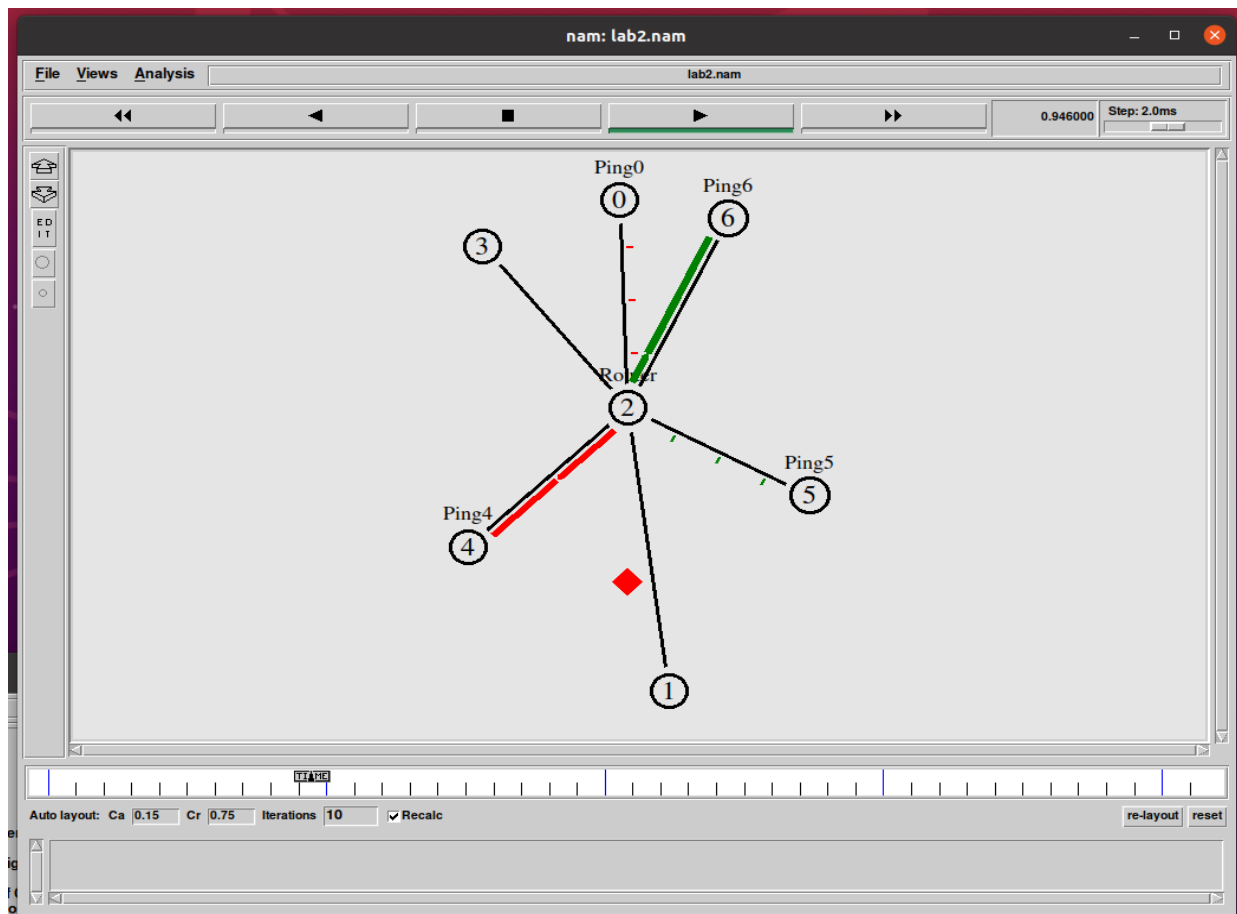
#Define a 'recv' function for the class 'Agent/Ping'

#The below function is executed when the ping agent receives a reply from the #destination

Agent/Ping instproc recv {from rtt} {

$self instvar node_

puts " The node [$node_ id] received an reply from $from with round trip time of $rtt"

}

proc finish {} {

global ns nf tf

exec nam lab4.nam &

exec echo "The total number of packet dropped due to congestion is:" &

exec grep -c "^d" lab4.tr &

$ns flush-trace

close $nf

close $tf

exit 0

}

#Schedule events

$ns at 0.1 "$ping0 send"

$ns at 0.2 "$ping0 send"

$ns at 0.3 "$ping0 send"

$ns at 0.4 "$ping0 send"

$ns at 0.5 "$ping0 send"

$ns at 0.6 "$ping0 send"

$ns at 0.7 "$ping0 send"

$ns at 0.8 "$ping0 send"

$ns at 0.9 "$ping0 send"

$ns at 1.0 "$ping0 send"

$ns at 1.1 "$ping0 send"

$ns at 1.2 "$ping0 send"

$ns at 1.3 "$ping0 send"

$ns at 1.4 "$ping0 send"

$ns at 1.5 "$ping0 send"

$ns at 1.6 "$ping0 send"

$ns at 1.7 "$ping0 send"

$ns at 1.8 "$ping0 send"

$ns at 0.1 "$ping5 send"

$ns at 0.2 "$ping5 send"

$ns at 0.3 "$ping5 send"

$ns at 0.4 "$ping5 send"

$ns at 0.5 "$ping5 send"

$ns at 0.6 "$ping5 send"

$ns at 0.7 "$ping5 send"

$ns at 0.8 "$ping5 send"

$ns at 0.9 "$ping5 send"

 $ns at 1.0 "$ping5 send"

$ns at 1.1 "$ping5 send"

$ns at 1.2 "$ping5 send"

$ns at 1.3 "$ping5 send"

$ns at 1.4 "$ping5 send"

$ns at 1.5 "$ping5 send"

$ns at 1.6 "$ping5 send"

$ns at 1.7 "$ping5 send"

$ns at 1.8 "$ping5 send"

$ns at 5.0 "finish"

$ns run

OUTPUT:

Program 3

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

Experiment Specific Instructions

1. To analyze the given problem you have to write a Tcl script and simulate withns2
2. Begin by specifying the trace files and the nam files to be created
3. Define a finish procedure
4. Determine and create the nodes that will be used to create the topology. Here inour experiment we are selecting 6 nodes namely 0, 1, 2, 3, 4, 5
5. Create the links to connect the nodes
6. Set up the LAN by specifying the nodes, and assign values for bandwidth, delay,queue type and channel to it
7. Optionally you can position and orient the nodes and links to view a nice videooutput with Nam
8. Set up the TCP and/or UDP connection(s) and the FTP/CBR (or any otherapplication) that will run over it
9. Schedule the different events like simulation start and stop, data transmissionstart and stop
10. Call the finish procedure and mention the time at what time your simulationwill end
11. Execute the script with ns

Steps for execution

- Open gedit editor and type program. Program name should have theextension " **.tcl**"
  - **[root@localhost ~]# gedit lab3.tcl / vi lab3.tcl**

- Save the program.
- Run the simulation program
  - **[root@localhost~]# ns lab6.tcl**
- To see the trace file contents open the file as,
  - **[root@localhost~]# gedit lab6.tr / vi lab6.tr**
- To see the graph contents open the file as,
  - **[root@localhost~]# xgraph congestion1.xg**
  - **[root@localhost~]# xgraph congestion2.xg**

Program

#set ns Simulator

set ns [new Simulator]

#define color for data flow

$ns color 1 Blue

$ns color 2 Red

#open trace file

set tracefile1 [open lab6.tr w]

set winfile [open winfile w]

$ns trace-all $tracefile1

#open namtrace file

set namfile [open lab6.nam w]

$ns namtrace-all $namfile

#define finish procedure

proc finish { } {

        global ns tracefile1 namfile

        $ns flush-trace

        close $tracefile1

        close $namfile

        exec nam lab6.nam &

        exit 0

        }

#create 6 nodes

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

set n4 [$ns node]

set n5 [$ns node]

$n1 shape box

#create link between nodes

$ns duplex-link $n0 $n2 2Mb 10ms DropTail

$ns duplex-link $n1 $n2 2Mb 10ms DropTail

$ns simplex-link $n2 $n3 0.3Mb 100ms DropTail

$ns simplex-link $n3 $n2 0.3Mb 100ms DropTail

set lan [$ns newLan "$n3 $n4 $n5" 0.5Mb 40ms LL Queue/DropTail MAC/802_3]

#give node position

$ns duplex-link-op $n0 $n2 orient right-down

$ns duplex-link-op $n1 $n2 orient right-up

$ns simplex-link-op $n3 $n2 orient left

$ns simplex-link-op $n2 $n3 orient right

#set queue size of link(n2-n3)

$ns queue-limit $n2 $n3 20

#setup tcp connection

set tcp [new Agent/TCP]

$ns attach-agent $n0 $tcp

set sink [new Agent/TCPSink]

$ns attach-agent $n4 $sink

$ns connect $tcp $sink

$tcp set fid_ 1

$tcp set packetSize_ 552

#set ftp over tcp connection

set ftp [new Application/FTP]

$ftp attach-agent $tcp

#setup a TCP1 connection

set tcp1 [new Agent/TCP]

$ns attach-agent $n1 $tcp1

set sink1 [new Agent/TCPSink]

$ns attach-agent $n5 $sink1

$ns connect $tcp1 $sink1

$tcp1 set fid_ 2

$tcp1 set packetSize_ 552

set telnet0 [new Application/Telnet]

$telnet0 attach-agent $tcp1

#title congestion window1

set outfile1 [open congestion1.xg w]

puts $outfile1 "TitleText: Congestion Window-- Source _tcp"

puts $outfile1 "xUnitText: Simulation Time(Secs)"

puts $outfile1 "yUnitText: Congestion WindowSize"

#title congestion window2

```
set outfile2 [open congestion2.xg w]
puts $outfile2 "TitleText: Congestion Window-- Source _tcp1"
puts $outfile2 "xUnitText: Simulation Time(Secs)"
puts $outfile2 "yUnitText: Congestion WindowSize"

proc plotWindow {tcpSource outfile} {
        global ns
        set time 0.1
        set now [$ns now]
        set cwnd [$tcpSource set cwnd_]
        puts $outfile "$now $cwnd"
        $ns at [expr $now+$time] "plotWindow $tcpSource $outfile"
        }
$ns at 0.1 "plotWindow $tcp $winfile"
$ns at 0.0 "plotWindow $tcp $outfile1"
$ns at 0.1 "plotWindow $tcp1 $outfile2"
$ns at 0.3 "$ftp start"
$ns at 0.5 "$telnet0 start"
$ns at 49.0 "$ftp stop"
$ns at 49.1 "$telnet0 stop"
$ns at 50.0 "finish"
$ns run
```
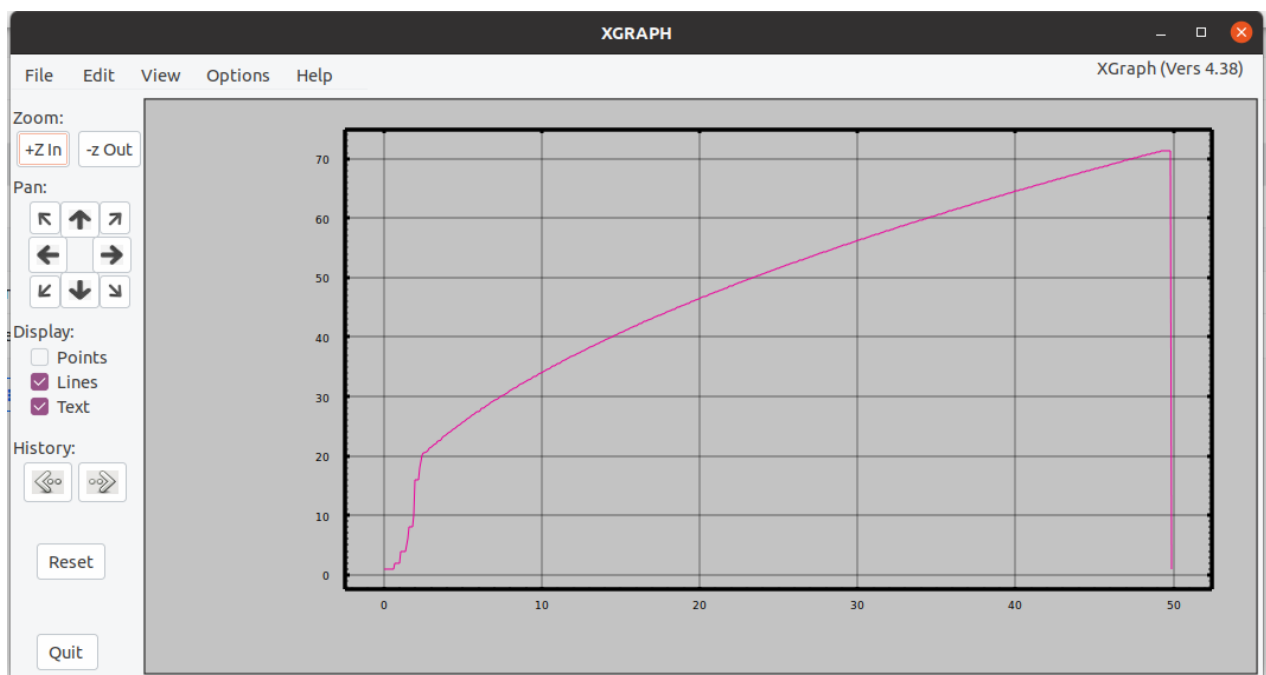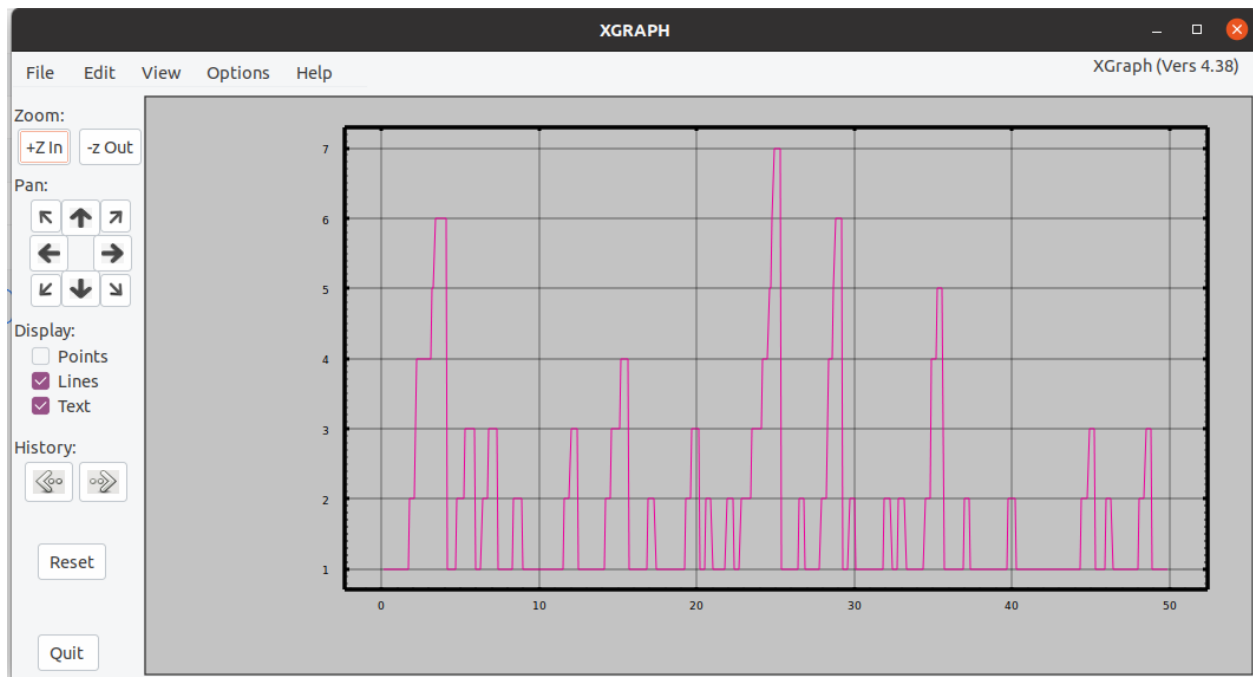
Topology:
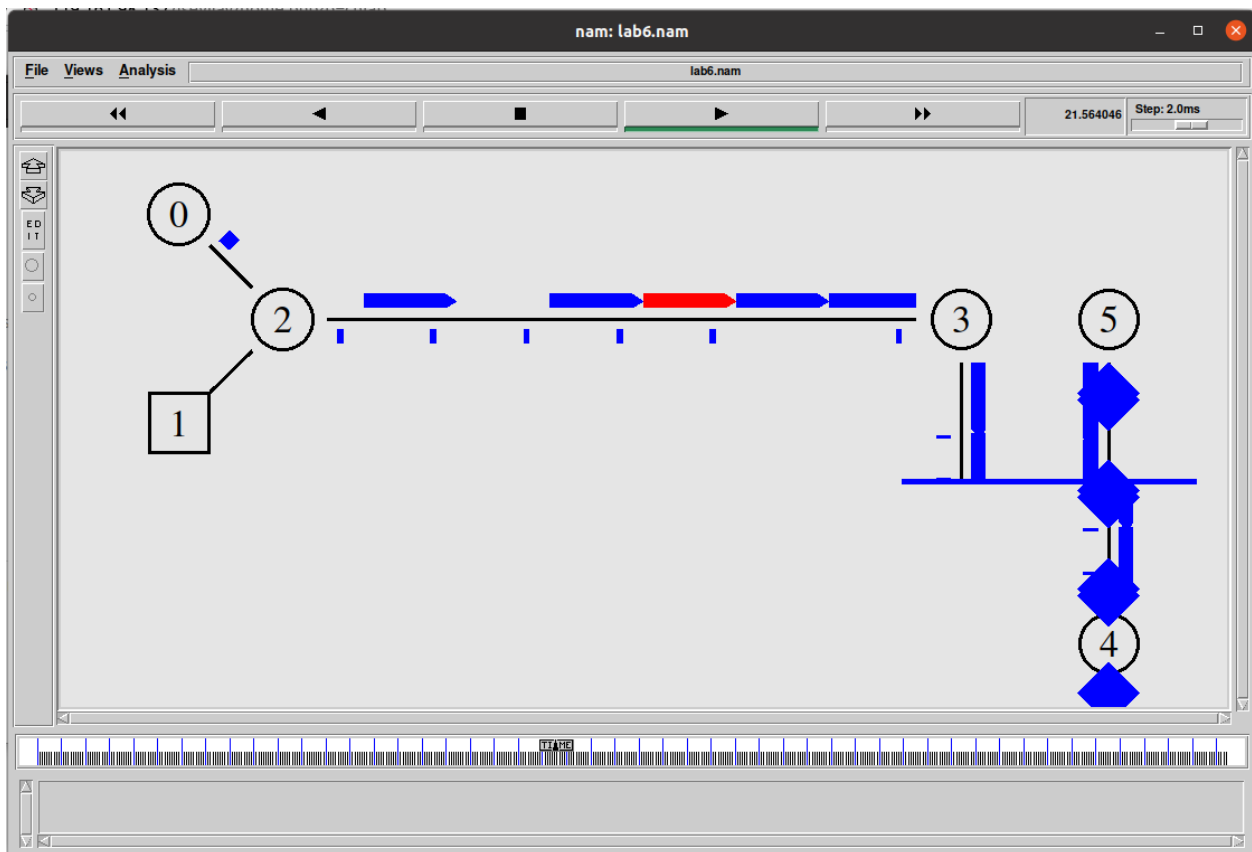
Congestion graph

ise@Ubuntu20:~/cnlab$ xgraph congestion1.xg

ise@Ubuntu20:~/cnlab$ xgraph congestion2.xg


```
  1 + 0.3 0 2 tcp 40 ------- 1 0.0 4.0 0 0
  2 - 0.3 0 2 tcp 40 ------- 1 0.0 4.0 0 0
  3 r 0.31016 0 2 tcp 40 ------- 1 0.0 4.0 0 0
  4 + 0.31016 2 3 tcp 40 ------- 1 0.0 4.0 0 0
  5 - 0.31016 2 3 tcp 40 ------- 1 0.0 4.0 0 0
  6 r 0.411227 2 3 tcp 40 ------- 1 0.0 4.0 0 0
  7 h 0.411227 3 6 tcp 40 ------- 1 0.0 4.0 0 0
  8 + 0.451227 3 6 tcp 40 ------- 1 0.0 4.0 0 0
  9 - 0.451227 3 6 tcp 40 ------- 1 0.0 4.0 0 0
 10 d 0.451231 5 6 tcp 40 ------- 1 0.0 4.0 0 0
 11 r 0.491231 6 4 tcp 40 ------- 1 0.0 4.0 0 0
 12 h 0.491231 4 6 ack 40 ------- 1 4.0 0.0 0 1
 13 + 0.531231 4 6 ack 40 ------- 1 4.0 0.0 0 1
 14 - 0.531231 4 6 ack 40 ------- 1 4.0 0.0 0 1
 15 d 0.531235 5 6 ack 40 ------- 1 4.0 0.0 0 1
 16 r 0.571235 6 3 ack 40 ------- 1 4.0 0.0 0 1
 17 + 0.571235 3 2 ack 40 ------- 1 4.0 0.0 0 1
 18 - 0.571235 3 2 ack 40 ------- 1 4.0 0.0 0 1
 19 r 0.672301 3 2 ack 40 ------- 1 4.0 0.0 0 1
 20 + 0.672301 2 0 ack 40 ------- 1 4.0 0.0 0 1
 21 - 0.672301 2 0 ack 40 ------- 1 4.0 0.0 0 1
 22 r 0.682461 2 0 ack 40 ------- 1 4.0 0.0 0 1
 23 + 0.682461 0 2 tcp 592 ------- 1 0.0 4.0 1 2
 24 - 0.682461 0 2 tcp 592 ------- 1 0.0 4.0 1 2
 25 + 0.682461 0 2 tcp 592 ------- 1 0.0 4.0 2 3
 26 - 0.684829 0 2 tcp 592 ------- 1 0.0 4.0 2 3
 27 r 0.694829 0 2 tcp 592 ------- 1 0.0 4.0 1 2
 28 + 0.694829 2 3 tcp 592 ------- 1 0.0 4.0 1 2
 29 - 0.694829 2 3 tcp 592 ------- 1 0.0 4.0 1 2
 30 r 0.697197 0 2 tcp 592 ------- 1 0.0 4.0 2 3
 31 + 0.697197 2 3 tcp 592 ------- 1 0.0 4.0 2 3
 32 - 0.710616 2 3 tcp 592 ------- 1 0.0 4.0 2 3
 33 r 0.810616 2 3 tcp 592 ------- 1 0.0 4.0 1 2
 34 h 0.810616 3 6 tcp 592 ------- 1 0.0 4.0 1 2
 35 r 0.826403 2 3 tcp 592 ------- 1 0.0 4.0 2 3
 36 h 0.826403 3 6 tcp 592 ------- 1 0.0 4.0 2 3
```
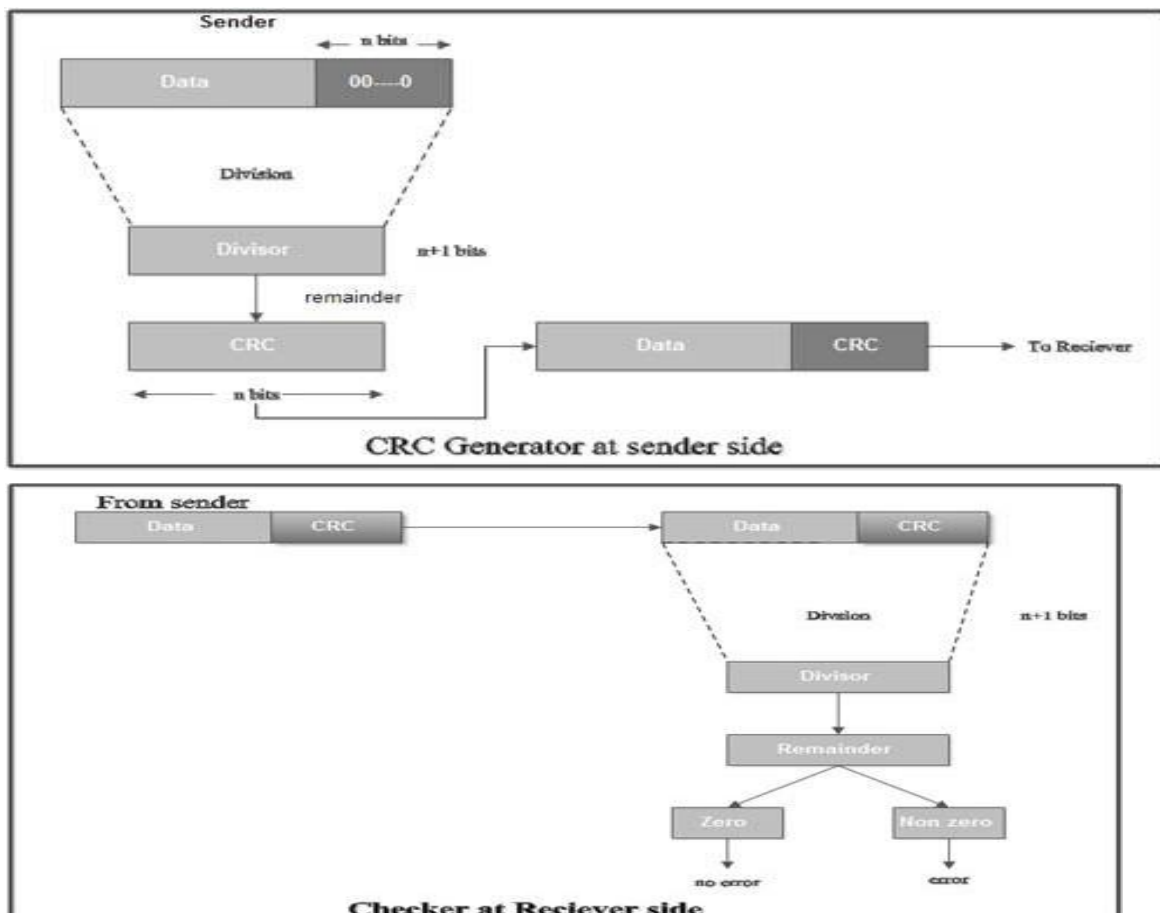
Program 4

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

A cyclic redundancy check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data.

Blocks of data entering these systems get a short check value attached, based on the remainder of a polynomial division of their contents. On retrieval the calculation is repeated, and corrective action can be taken against presumed data corruption if the check values do not match.

Algorithm:-

1. 1. Given a bit string, append 0S to the end of it (the number of 0s 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.

CRC Generator at sender side

Checker at Reciever side

Program:

```java
package cnlab;
import java.io.*;
import java.util.Scanner;
class Crc
{
    public static void main(String args[])
    {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter message bits: ");
        String message = sc.nextLine();
        System.out.print("Enter generator: ");
        String generator = sc.nextLine();
        int data[] = new int[message.length() +
generator.length() - 1];
        int divisor[] = new int[generator.length()];
        for(int i=0;i<message.length();i++)
        data[i] = Integer.parseInt(message.charAt(i)+"");
        for(int i=0;i<generator.length();i++)
            divisor[i] =
Integer.parseInt(generator.charAt(i)+"");
        for(int i=0;i<message.length();i++)
            {
            if(data[i]==1)
                for(int j=0;j<divisor.length;j++) data[i+j] ^=
divisor[j];
            }
        System.out.print("The checksum code is: ");
        for(int i=0;i<message.length();i++)
            data[i] = Integer.parseInt(message.charAt(i)+"");
        for(int i=0;i<data.length;i++)
            System.out.print(data[i]);
        System.out.println();
        System.out.print("Enter checksum code: ");
        message = sc.nextLine();
```

```java
            System.out.print("Enter generator: ");
            generator = sc.nextLine();
            data = new int[message.length() + generator.length() -
1];
            divisor = new int[generator.length()];
            for(int i=0;i<message.length();i++)
                data[i] = Integer.parseInt(message.charAt(i)+"");
            for(int i=0;i<generator.length();i++)
                divisor[i] =
Integer.parseInt(generator.charAt(i)+"");
            for(int i=0;i<message.length();i++)
                {
                if(data[i]==1)
                    for(int j=0;j<divisor.length;j++) data[i+j] ^=
divisor[j];
                }
            boolean valid = true;
            for(int i=0;i<data.length;i++)
                if(data[i]==1)
                    {
                        valid = false;
                        break;
                    }
            if(valid==true)
                System.out.println("Data stream is valid");
            else
                System.out.println("Data stream is invalid. CRC
error occurred.");
        }
}
```

OUTPUT:

Enter message bits: 101001

Enter generator: 1101

The checksum code is: 101001001

Enter checksum code: 101001001

Enter generator: 1101

Data stream is valid

Enter message bits: 101001

Enter generator: 1101

The checksum code is: 101001001

Enter checksum code: 101101001

Enter generator: 1101

Data stream is invalid. CRC error occurred.

Program 5

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

ALGORITHM:

1. Start the program.
2. Get the frame size from the user
3. To create the frame based on the user request.
4. To send frames to server from the client side.
5. If your frames reach the server it will send ACK signal to client otherwise it will send NACK signal to client.
6. Stop the program

Program:

Slide Sender Code:

```java
package pgm5;
import java.net.*;
import java.io.*;
import java.rmi.*;
public class slidsender
{
@SuppressWarnings({ "deprecation", "deprecation", "deprecation" })
public static void main(String a[])throws Exception
{
ServerSocket ser=new ServerSocket(4000);
Socket s=ser.accept();
DataInputStream in=new DataInputStream(System.in);
DataInputStream in1=new DataInputStream(s.getInputStream());
String sbuff[]=new String[8];
PrintStream p;
int sptr=0,sws=8,nf,ano,i; String ch;
do
{
p=new PrintStream(s.getOutputStream());
System.out.print("Enter the no. of frames : ");
nf=Integer.parseInt(in.readLine());
```

```
p.println(nf);
if(nf<=sws-1)
{
System.out.println("Enter "+nf+" Messages to be send\n");
for(i=1;i<=nf;i++)
{
sbuff[sptr]=in.readLine();
p.println(sbuff[sptr]);
sptr=++sptr%8;
}
sws-=nf;
System.out.print("Acknowledgment received");
ano=Integer.parseInt(in1.readLine());
System.out.println(" for "+ano+" frames");
sws+=nf;
}
else
{
System.out.println("The no. of frames exceeds window size");
break;
}
System.out.print("\nDo you wants to send some more frames : ");
ch=in.readLine();
p.println(ch);
}
while(ch.equals("yes"));
s.close();
}
}
```

Slide Receiver Code:

```
package pgm5;
import java.net.*;
import java.io.*;
public class slidreceiver
{
```

```java
@SuppressWarnings("deprecation")
public static void main(String a[])throws Exception
{
Socket s=new Socket(InetAddress.getLocalHost(),4000);
DataInputStream in=new DataInputStream(s.getInputStream());
PrintStream p=new PrintStream(s.getOutputStream());
int i=0,rptr=-1,nf,rws=8;
String rbuf[]=new String[8];
String ch;
System.out.println();
do
{
nf=Integer.parseInt(in.readLine());
if(nf<=rws-1)
{
for(i=1;i<=nf;i++)
{
rptr=++rptr%8;
rbuf[rptr]=in.readLine();
System.out.println("The received Frame " +rptr+" is : "+rbuf[rptr]);
}
rws-=nf;
System.out.println("\nAcknowledgment sent\n");
p.println(rptr+1);
rws+=nf;
}
else
break;
ch=in.readLine();
}
while(ch.equals("yes"));
}
}
```

OUTPUT : 1

Enter the no. of frames : 4

Enter 4 Messages to be send

welcome to CN Lab

sliding window protocol

thank you

bye

Acknowledgment received for 4 frames

Do you wants to send some more frames : no

The received Frame 0 is : welcome to CN Lab

The received Frame 1 is : sliding window protocol

The received Frame 2 is : thank you

The received Frame 3 is : bye

Acknowledgment sent

OUTPUT : 2

Enter the no. of frames : 1
Enter 1 Messages to be send
Welcome to ISE Department
Acknowledgment received for 1 frames
Do you wants to send some more frames : yes
Enter the no. of frames : 1
Enter 1 Messages to be send
bye
Acknowledgment received for 2 frames
Do you wants to send some more frames : no
The received Frame 0 is : Welcome to ISE Department
Acknowledgment sent
The received Frame 1 is : bye
Acknowledgment sent

Program 6

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

Program:

```java
package cnlab;

import java.util.Scanner;

public class ford
{
    private int D[];
    private int num_ver;
    public static final int MAX_VALUE = 999;
    public ford(int num_ver)
    {
        this.num_ver = num_ver;
        D = new int[num_ver + 1];
    }
public void BellmanFordEvaluation(int source, int A[][])
{
    for (int node = 1; node <= num_ver; node++)
        {
        D[node] = MAX_VALUE;
        }
    D[source] = 0;
    for (int node = 1; node <= num_ver - 1; node++)
        {
        for (int sn = 1; sn <= num_ver; sn++)
            {
            for (int dn = 1; dn <= num_ver; dn++)
                {
                if (A[sn][dn] != MAX_VALUE)
                    {
                    if (D[dn] > D[sn]+ A[sn][dn])
                        D[dn] = D[sn] + A[sn][dn];
                    }
                }
```

```
            }
        }
    for (int sn = 1; sn <= num_ver; sn++)
        {
        for (int dn = 1; dn <= num_ver; dn++)
            {
            if (A[sn][dn] != MAX_VALUE)
                {
                if (D[dn] > D[sn]+ A[sn][dn])
                    System.out.println("The Graph contains
negative egde cycle");
                }
            }
        }
    for (int vertex = 1; vertex <= num_ver; vertex++)
        {
        System.out.println("distance of source "+source+" to
"+vertex+" is " + D[vertex]);
        }
    }
public static void main(String[ ] args)
{
    int num_ver = 0; int 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;
```

```java
                continue;
                }
            if (A[sn][dn] == 0)
                {
                A[sn][dn] = MAX_VALUE;
                }
            }
        }
    System.out.println("Enter the source vertex");
    source = scanner.nextInt();
    ford b = new ford (num_ver);
    b.BellmanFordEvaluation(source, A);
    scanner.close();
    }
}
```

OUTPUT:

```
Enter the number of vertices
5
Enter the adjacency matrix
0 3 2 4 999
3 0 999 2 999
2 999 0 2 3
4 2 2 0 4
999 999 3 4 0
Enter the source vertex
1
distance of source 1 to 1 is 0
distance of source 1 to 2 is 3
distance of source 1 to 3 is 2
distance of source 1 to 4 is 4
distance of source 1 to 5 is 5
```
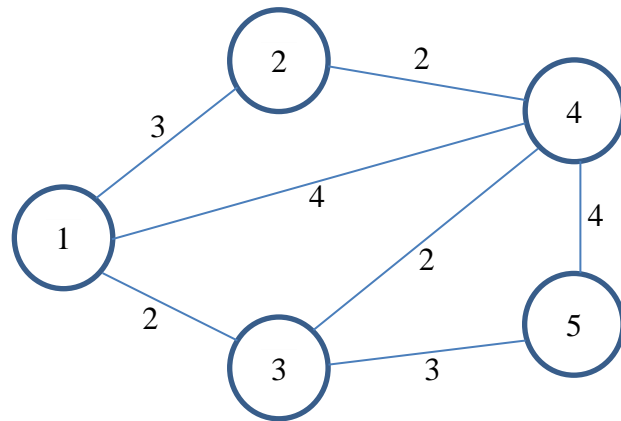
```
Enter the number of vertices
6
Enter the adjacency matrix
0     5     999   999   999   999
999   0     1     2     999   999
999   999   0     999   1     999
999   999   999   0     999   2
999   999   999   -1    0     999
999   999   999   999   -3    0
Enter the source vertex
1
The Graph contains negative egde cycle
The Graph contains negative egde cycle
distance of source 1 to 1 is 0
distance of source 1 to 2 is 5
distance of source 1 to 3 is 6
distance of source 1 to 4 is 2
distance of source 1 to 5 is 2
distance of source 1 to 6 is 5
```
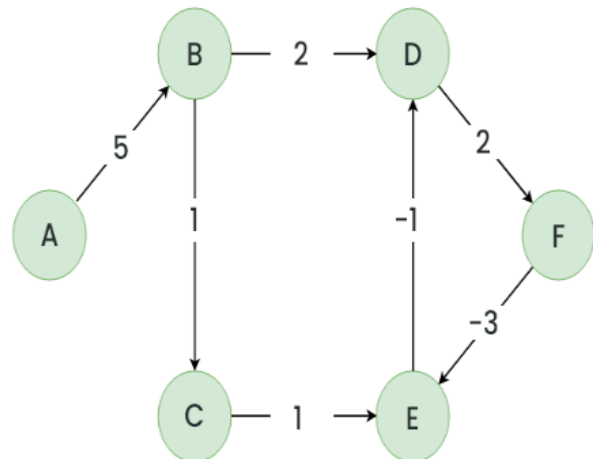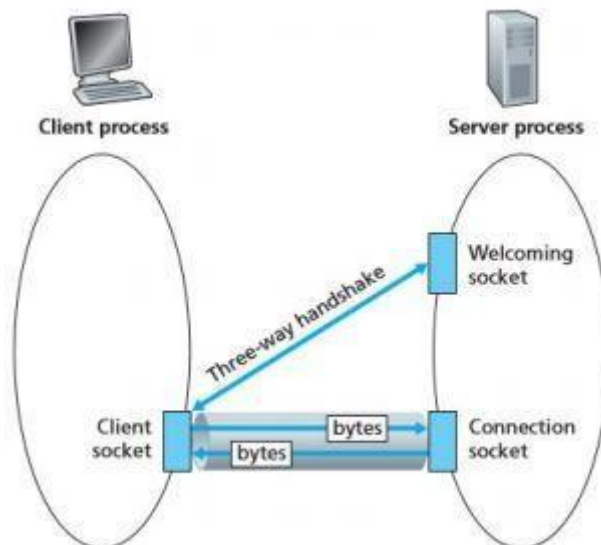
Program 7

Using TCP/IP sockets, write a client – server program to make the client send the file name and to make the server send back the contents of the requested file if present.

Algorithm (Client Side)

1. Start.
2. Create a socket using socket() system call.
3. Connect the socket to the address of the server using connect() system call.
4. Send the filename of required file using send() system call.
5. Read the contents of the file sent by server by recv() system call.
6. Stop.

Algorithm (Server Side)

1. Start.
2. Create a socket using socket() system call.
3. Bind the socket to an address using bind() system call.
4. Listen to the connection using listen() system call.
5. accept connection using accept()
6. Receive filename and transfer contents of file with client.
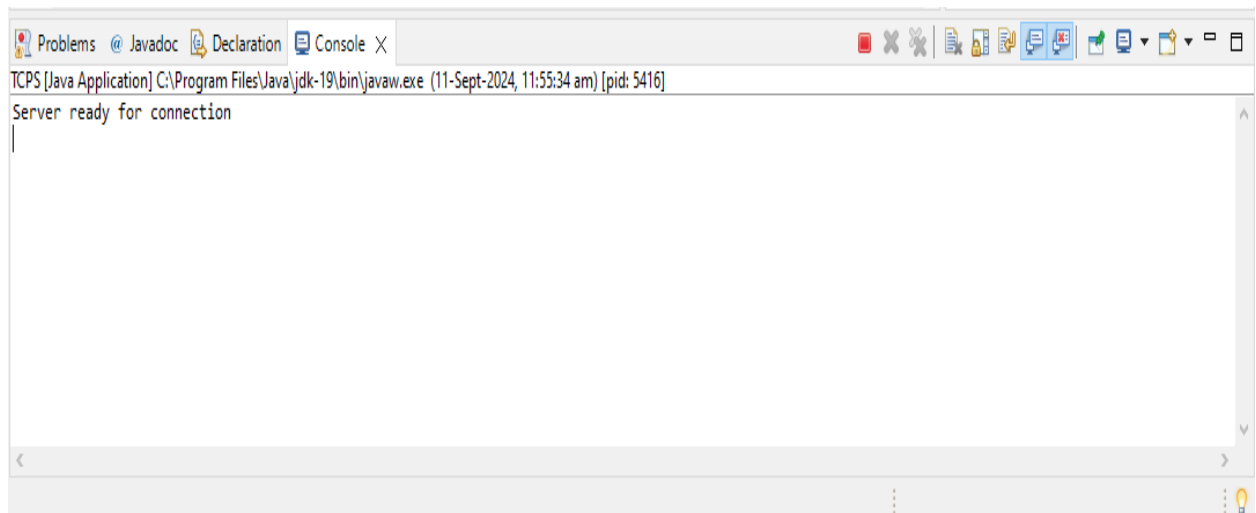7. Stop.

CLIENT SIDE:

```java
package pgm7;
import java.net.*;
import java.io.*;
public class TCPC
{
    public static void main(String[] args) throws Exception
    {
        Socket sock=new Socket("127.0.01",4000);
        System.out.println("Enter the filename");
        BufferedReader keyRead=new BufferedReader(new
InputStreamReader(System.in));
        String fname=keyRead.readLine();
        OutputStream ostream=sock.getOutputStream();
        PrintWriter pwrite=new PrintWriter(ostream,true);
        pwrite.println(fname);
        InputStream istream=sock.getInputStream();
        BufferedReader socketRead=new BufferedReader(new
InputStreamReader(istream));
        String str;
        while((str=socketRead.readLine())!=null)
        {
            System.out.println(str);
        }
        pwrite.close();
        socketRead.close();
        keyRead.close();
    }
}
```
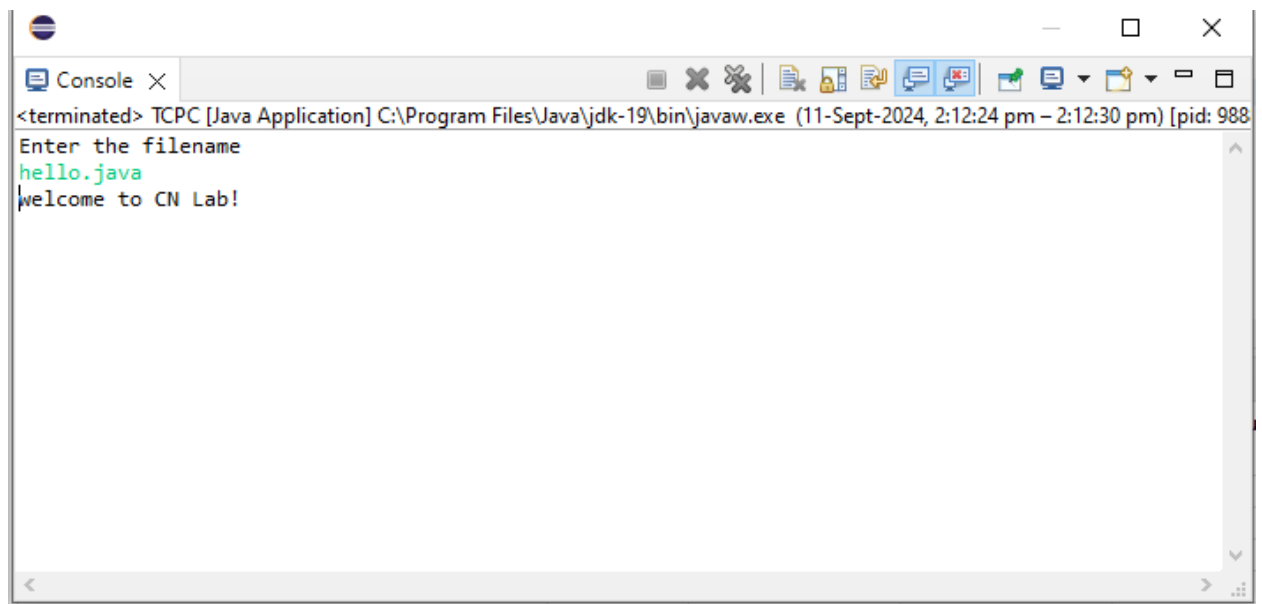
SERVER SIDE:

```java
package pgm7;
import java.net.*;
import java.io.*;
public class TCPS
{
    public static void main(String[] args) throws Exception
    {
        ServerSocket sersock=new ServerSocket(4000);
        System.out.println("Server ready for connection");
        Socket sock=sersock.accept();
        System.out.println("Connection Is successful and waiting for chatting");
        InputStream istream=sock.getInputStream();
        BufferedReader fileRead=new BufferedReader(new InputStreamReader(istream));
        String fname=fileRead.readLine();
        BufferedReader ContentRead=new BufferedReader(new FileReader(fname));
        OutputStream ostream=sock.getOutputStream();
        PrintWriter pwrite=new PrintWriter(ostream,true);
        String str;
        while((str=ContentRead.readLine())!=null)
        {
            pwrite.println(str);
        }
        sock.close();
        sersock.close();
        pwrite.close();
        fileRead.close();
        ContentRead.close();
    }
}
```

OUTPUT:

Program 8

Develop a program on a datagram socket for client/server to display the messages on client side, typed at the server side.
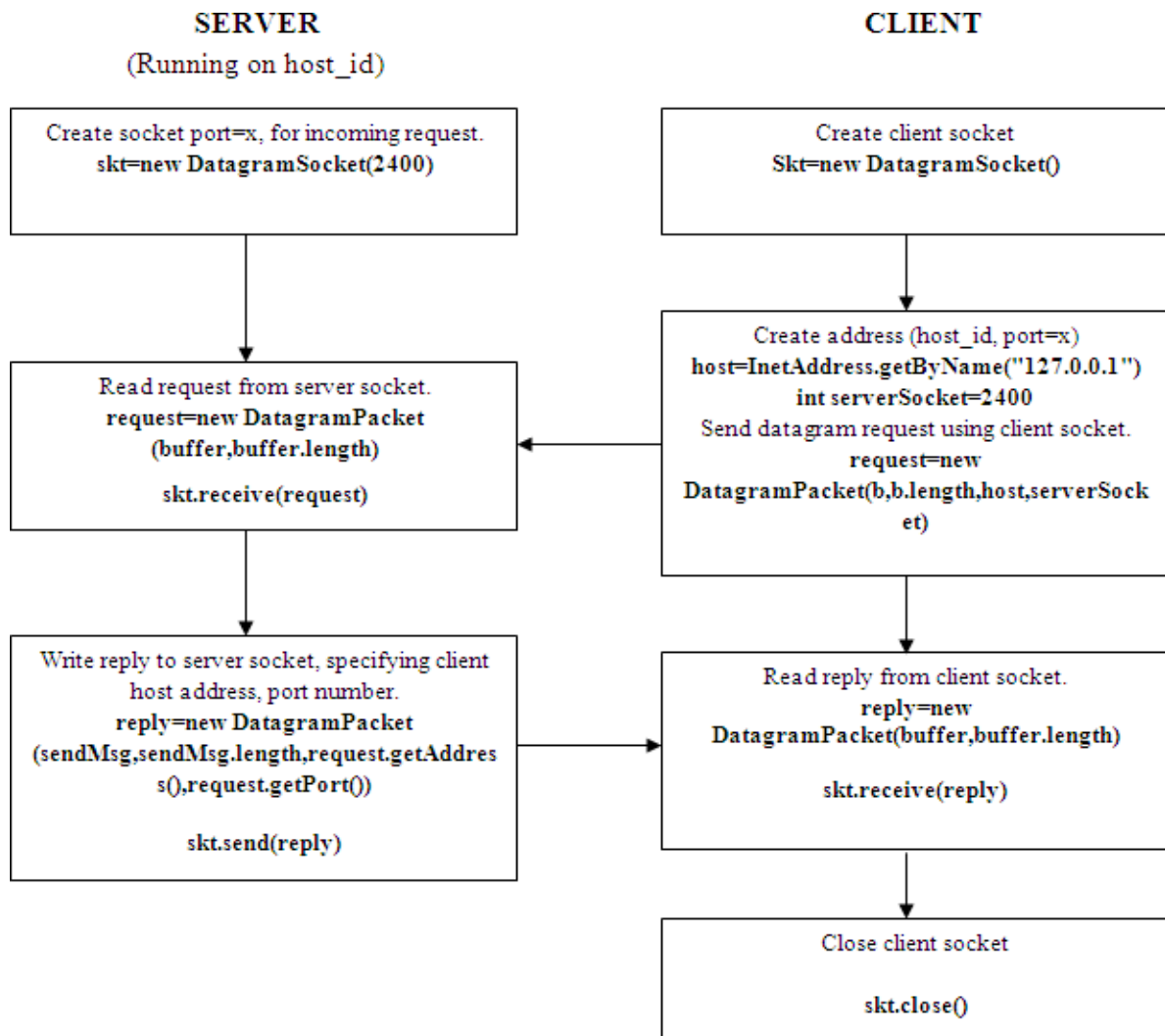
UDP client/server communication flow.

Methods and description

- **DatagramSocket(int port) throws SocketEeption:** it creates a datagram socket and binds it with the given Port Number.
- **DatagramPacket(byte[] buffer, int length):** it creates a datagram packet. This constructor is used to receive the packets.
- **DatagramPacket(byte[] buffer, int length, InetAddress address, int port):** it creates a datagram packet. This constructor is used to send the packets.
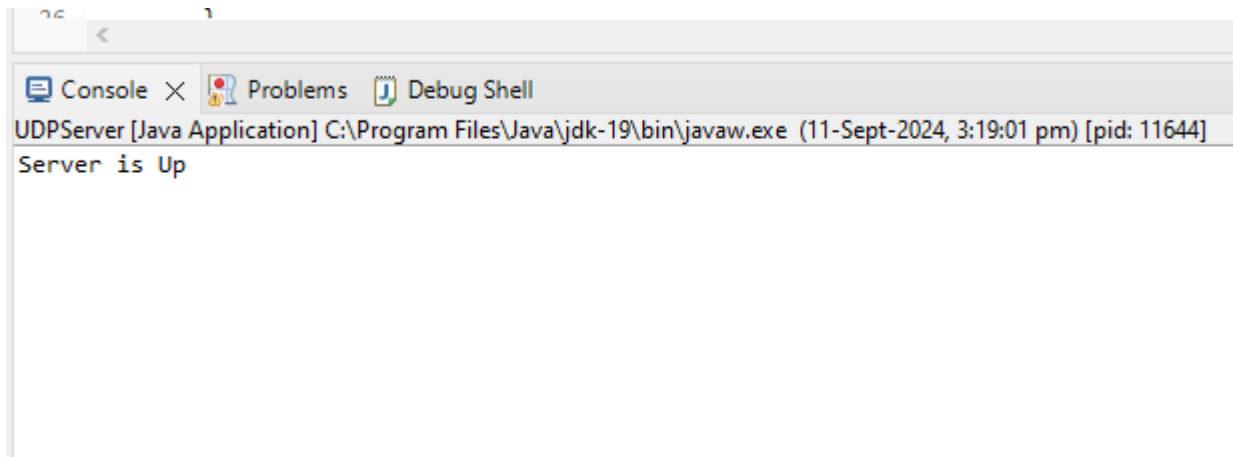
Program:

Client Side:

```java
package pgm8;
import java.io.*;
import java.net.*;
import java.net.InetAddress;
class UDPClient
{
    public static void main(String[] args)throws Exception
    {
        BufferedReader inFromUser=new BufferedReader(new
InputStreamReader(System.in));
        DatagramSocket clientSocket=new DatagramSocket();
        InetAddress
IPAddress=InetAddress.getByName("localhost");
        byte[] sendData=new byte[1024];
        byte[] receiveData=new byte[1024];
        System.out.println("Enter the sting to be converted in
to Upper case");
        String sentence=inFromUser.readLine();
        sendData=sentence.getBytes();
        DatagramPacket sendPacket=new

    DatagramPacket(sendData,sendData.length,IPAddress,9876);
        clientSocket.send(sendPacket);
        DatagramPacket receivePacket=new
DatagramPacket(receiveData,receiveData.length);
        clientSocket.receive(receivePacket);
        String modifiedSentence=new
String(receivePacket.getData());
        System.out.println("FROM SERVER:"+modifiedSentence);
        clientSocket.close();
    }
}
```
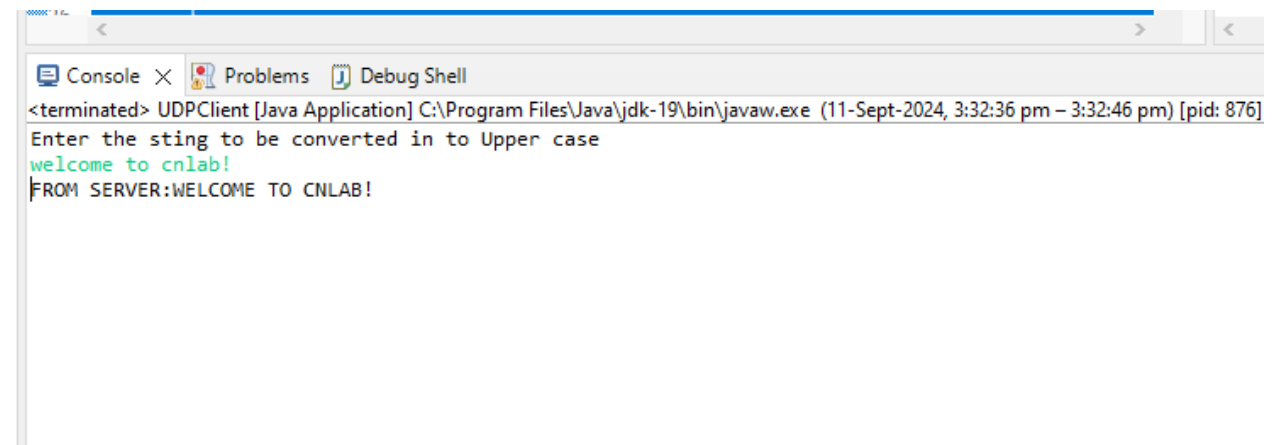
<u>Server Side:</u>

```java
package pgm8;
import java.net.*;
import java.net.InetAddress;
class UDPServer
{
    public static void main(String args[])throws Exception
    {
        DatagramSocket serverSocket = new DatagramSocket(9876);
        byte[] receiveData=new byte[1024];
        byte[] sendData=new byte[1024];
        while(true)
        {
            System.out.println("Server is Up");
            DatagramPacket receivePacket=new

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

DatagramPacket(sendData,sendData.length,IPAddress,port);
            serverSocket.send(sendPacket);
        }
    }
}
```

OUTPUT:

Console X Problems Debug Shell

UDPServer [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (11-Sept-2024, 3:19:01 pm) [pid: 11644]

Server is Up

Console X Problems Debug Shell

<terminated> UDPClient [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (11-Sept-2024, 3:32:36 pm – 3:32:46 pm) [pid: 876]

Enter the sting to be converted in to Upper case
welcome to cnlab!
FROM SERVER:WELCOME TO CNLAB!

Program 9

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

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 $N = P$ x Q
3. Compute $Z = (P\text{-}1)$ x $(Q\text{-}1)$.
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 x $D \equiv 1$ (mod 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 x Q = 11 x 3 = 33

   Z = (P-1) x (Q-1) = 10 x 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

Program:

```java
package pgm9;

import java.io.DataInputStream;
import java.io.IOException;
import java.math.BigInteger;
import java.util.Random;

public class RSA
{
    private BigInteger p,q,N,phi,e,d;
    private int       bitlength = 48;
    private Random     r;

    public RSA()
    {
        r = new Random();
        p = BigInteger.probablePrime(bitlength, r);
        q = BigInteger.probablePrime(bitlength, r);
        System.out.println("Prime number p is"+p);
      System.out.println("prime number q is"+q);
        N = p.multiply(q);
        phi =
p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
        e = BigInteger.probablePrime(bitlength / 2, r);
        while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 &&
e.compareTo(phi) < 0)
        {
            e.add(BigInteger.ONE);
        }
        System.out.println("Public key is"+e);
        d = e.modInverse(phi);
        System.out.println("Private key is"+d);
    }

    public RSA(BigInteger e, BigInteger d, BigInteger N)
```

```java
    {
        this.e = e;
        this.d = d;
        this.N = N;
    }



    public static void main(String[] args) throws IOException
    {
        RSA rsa = new RSA();
        DataInputStream in = new DataInputStream(System.in);
        String teststring;
        System.out.println("Enter the plain text:");
        teststring = in.readLine();
        System.out.println("Encrypting String: " + teststring);
        System.out.println("String in Bytes: " +
bytesToString(teststring.getBytes()));
        // encrypt
        byte[] encrypted = rsa.encrypt(teststring.getBytes());
        // decrypt
        byte[] decrypted = rsa.decrypt(encrypted);
        System.out.println("Decrypting Bytes: " +
bytesToString(decrypted));
        System.out.println("Decrypted String: " + new
String(decrypted));
    }


    private static String bytesToString(byte[] encrypted)
    {
        String test = "";
        for (byte b : encrypted)
        {
            test += Byte.toString(b);
        }
        return test;
    }
```

```java
// Encrypting message
public byte[] encrypt(byte[] message)
{

 return (new BigInteger(message)).modPow(e, N).toByteArray();

}


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


}
}
```

OUTPUT:

Prime number p is253073860231303

prime number q is258382793336351

Public key is14869111

Private key is2822987007256696875375113691

Enter the plain text:

cnlab

Encrypting String: cnlab

String in Bytes: 991101089798

Decrypting Bytes: 991101089798

Decrypted String: cnlab

Prime number p is595801061459120507196527000059

Prime number q is440063628626284523574325506297

Public key is18279081972809

Private key
is735127728707513965980832424745546751479161839555905807 2
57

Enter the plain text:

RSA Algorithm

Encrypting String: RSA Algorithm

String in Bytes: 8283653265108103111141051161041 01109

Decrypting Bytes: 8283653265108103111141051161041 01109

Decrypted String: RSA Algorithm

Program 10

Develop a program for congestion control using a 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.
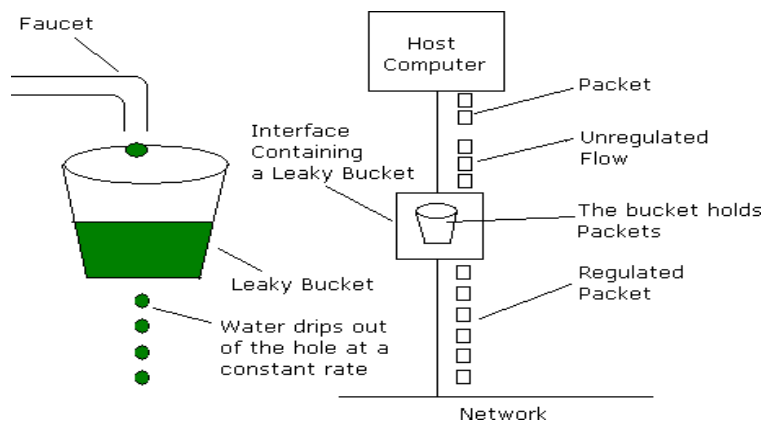
Program:

```java
package cnlab;
import java.util.Scanner;
public class Leakybucket
{
    public static void main(String [] args)
    {
        int i;
        int a[]=new int[20];
        int buck_rem=0, buck_cap=4, rate=3, sent, recv;
        Scanner in=new Scanner(System.in);
        System.out.println("Enter the number of Packets");
        int n=in.nextInt();
        System.out.println("Enter the Packets");
        for(i=1;i<=n;i++)
            a[i]=in.nextInt();
        System.out.println("Clock \t Packet size \t accept \t sent \t remaining");
        for(i=1;i<=n;i++)
        {
            if(a[i]!=0)
            {
                if(buck_rem+a[i]>buck_cap)
                    recv=-1;
                else
                {
```

```java
                recv=a[i];
                buck_rem+=a[i];
            }
        }
        else
            recv=0;
        if(buck_rem!=0)
        {
            if(buck_rem<rate)
            {
                sent=buck_rem;
                buck_rem=0;
            }
            else
            {
                sent=rate;
                buck_rem=buck_rem-rate;
            }
        }
        else
            sent=0;
        if(recv==-1)
            System.out.println(+i+ "\t\t" +a[i]+ "\t
dropped \t" + sent + "\t" +buck_rem);
        else
            System.out.println(+i+ "\t\t" +a[i] +"\t\t"
+recv +"\t" +sent +"\t" +buck_rem);
        }
    }
}
```

OUTPUT:

```
Enter the number of packets
3
Enter the packets
4
3
2
```

Clock	packet size	accept	sent	remaining
1	4	4	3	1
2	3	3	3	1
3	2	2	3	0

```
Enter the number of packets
4
Enter the packets
4
3
2
5
```

Clock	packet size	accept	sent	remaining
1	4	4	3	1
2	3	3	3	1
3	2	2	3	0
4	5	dropped	0	0