



DEPARTMENT OF CSE- ARTIFICIAL INTELLIGENCE

LAB MANUAL

**Computer Network Laboratory
Manual
(22BCS502)
(V SEMESTER)**

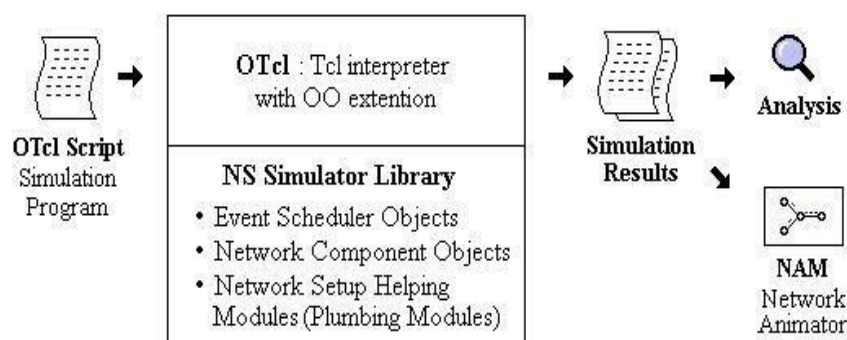
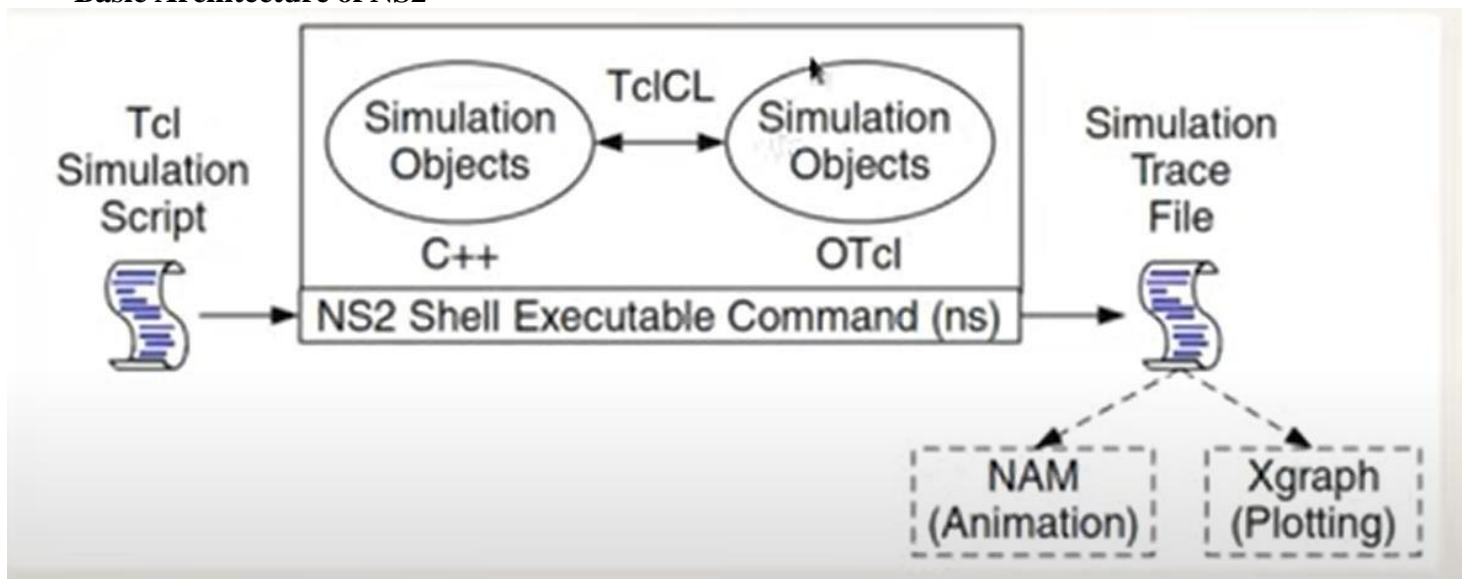
**Prepared by
Prof. A.N. Karthikeyan
Assistant Professor
Dept.of CSE-AI, SVCE.**

Introduction to NS-2

NS2 is an open-source simulation tool that runs on Linux. It is a discrete 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 `tcsh`; after a command completes, `tcsh` 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, MAC etc
- Error Models: Uniform, bursty etc
- Misc: Radio propagation, Mobility models , Energy Models
- Topology Generation tools
- Visualization tools (NAM), Tracing

Structure of NS

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

Platforms

It can be employed in most unix systems(FreeBSD, Linux, Solaris) and Windows.

Source code

Most of NS2 code is in C++

Scripting language

It uses TCL as its scripting language OTcl adds object orientation to TCL.

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 control 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.

PART A**Experiment No:1****THREE NODE POINT TO POINT NETWORK**

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

Program:

```
set ns [new Simulator]
set f [open lab1.tr w]
$ns trace-all $f
set nf [open lab1.nam w]
$ns namtrace-all $nf

proc finish {} {
    global f nf ns
    $ns flush-trace
    close $f
    close $nf
    exec nam lab1.nam &
    exit 0
}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]

$ns label "TCP Source"
$ns label "UDP Source"
$ns label "Sink"
$ns color 1 red
$ns color 2 yellow

$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 20ms DropTail
$ns queue-limit $n1 $n2 10
$ns duplex-link-op $n0 $n1 orient right
$ns duplex-link-op $n1 $n2 orient right

set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005

set null0 [new Agent/Null]
$ns attach-agent $n2 $null0
$ns connect $udp0 $null0
```

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

```
set sink [new Agent/TCPSink]
$ns attach-agent $n2 $sink
$ftp0 set maxPkts_ 1000
$ns connect $tcp0 $sink
$udp0 set class_ 1
$tcp0 set class_ 2
```

```
$ns at 0.1 "$cbr0 start"
$ns at 1.0 "$ftp0 start"
$ns at 4.0 "$ftp0 stop"
$ns at 4.5 "$cbr0 stop"
$ns at 5.0 "finish"
$ns run
```

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.
- Run the simulation program
[root@localhost ~]# ns lab1.tcl
- Here “ns” indicates network simulator. We get the topology shown in the network animator. Now press the play button in the simulation window and the simulation will begin.
- To calculate the number of packet dropped execute the following command.
[root@localhost ~]# grep ^d lab1.tr | grep “cbr” -c
[root@localhost ~]# grep ^d lab1.tr | grep “tcp” -c
- Write the value of number of packet dropped in observation sheet. Repeat the above step by changing the bandwidth to [0.5Mb, 1Mb, 1.5Mb, 2Mb] to the following line of the program.
\$ns duplex-link \$n0 \$n1 0.5Mb 10ms DropTail
\$ns duplex-link \$n1 \$n2 0.5Mb 20ms DropTail
- Plot a graph with x- axis with bandwidth and y-axis with number of packet dropped of TCP and UDP.

Experiment No: 2

TRANSMISSION OF PING MESSAGE

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

```
set ns [new Simulator]
set f [open lab2.tr w]
$ns trace-all $f
set nf [open lab2.nam w]
$ns namtrace-all $nf

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

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

```
$n0 label "ping0"
$n1 label "ping1"
$n2 label "R1"
$n3 label "R2"
$n4 label "ping4"
$n5 label "ping5"
```

```
$ns color 1 red
$ns color 2 blue
$ns color 3 green
$ns color 4 orange
```

```
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 0.5Mb 30ms DropTail
$ns duplex-link $n3 $n4 1Mb 10ms DropTail
$ns duplex-link $n3 $n5 1Mb 10ms DropTail
```

```
set ping0 [new Agent/Ping]
$ns attach-agent $n0 $ping0
set ping1 [new Agent/Ping]
$ns attach-agent $n1 $ping1
set ping4 [new Agent/Ping]
$ns attach-agent $n4 $ping4
set ping5 [new Agent/Ping]
$ns attach-agent $n5 $ping5
$ns connect $ping0 $ping4
$ns connect $ping1 $ping5
```

```

proc sendPingPacket {} {
global ns ping0 ping1 ping4 ping5
set intervalTime 0.001
set now [$ns now]
$ns at [expr $now + $intervalTime] "$ping0 send"
$ns at [expr $now + $intervalTime] "$ping1 send"
$ns at [expr $now + $intervalTime] "$ping4 send"
$ns at [expr $now + $intervalTime] "$ping5 send"
$ns at [expr $now + $intervalTime] "sendPingPacket"
}

```

```

Agent/Ping instproc recv {from rtt} {
global seq
$self instvar node_
puts "The node [$node_id] received an ACK from the node $from
with RTT $rtt ms"
}

```

```

$ping0 set class_ 1
$ping1 set class_ 2
$ping4 set class_ 4
$ping5 set class_ 5
$ns at 0.01 "sendPingPacket"
$ns at 10.0 "finish"
$ns run

```

Steps for execution:

- Open vi editor and type program. Program name should have the extension “.tcl ”
[root@localhost ~]# vi lab2.tcl
- 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 lab2.tcl
- Here “ns” indicates network simulator. We get the topology shown in the network animator. Now press the play button in the simulation window and the simulation will begin.
- To calculate the number of packet dropped. Execute the following command.
[root@localhost~]#grep ^d lab2.tr -c
- Write the value of number of packet sent of TCP and UDP in observation sheet. Repeat the above step by changing the bandwidth to [0.5Mb, 1Mb,1.5Mb, 2Mb]to the following line of the program.

```

$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 0.5Mb 30ms DropTail
$ns duplex-link $n3 $n4 1Mb 10ms DropTail
$ns duplex-link $n3 $n5 1Mb 10ms DropTail

```
- Plot a graph with x- axis with bandwidth and y-axis with number of packet dropped due to ping.

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

Aim: *Simulate an Ethernet LAN using 'n' nodes and set multiple traffic nodes and plot congestion window for different source / destination.*

Program:

lab3.tcl

```
set ns [new Simulator]
set tf [open lab3.tr w]
$ns trace-all $tf
set nf [open lab3.nam w]
$ns namtrace-all $nf
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
$n0 color "magenta"
$n0 label "src1"
$n2 color "magenta"
$n2 label "src2"
$n3 color "blue"
$n3 label "dest2"
$n5 color "blue"
$n5 label "dest1"
$ns make-lan "$n0 $n1 $n2 $n3 $n4" 100Mb 100ms LL Queue/DropTail
Mac/802_3
$ns duplex-link $n4 $n5 1Mb 1ms DropTail
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ftp0 set packetSize_ 500
$ftp0 set interval_ 0.0001
set sink5 [new Agent/TCPSink]
$ns attach-agent $n5 $sink5
$ns connect $tcp0 $sink5
set tcp2 [new Agent/TCP]
$ns attach-agent $n2 $tcp2
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2

$ftp2 set packetSize_ 600
$ftp2 set interval_ 0.001
set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3
$ns connect $tcp2 $sink3
```

```

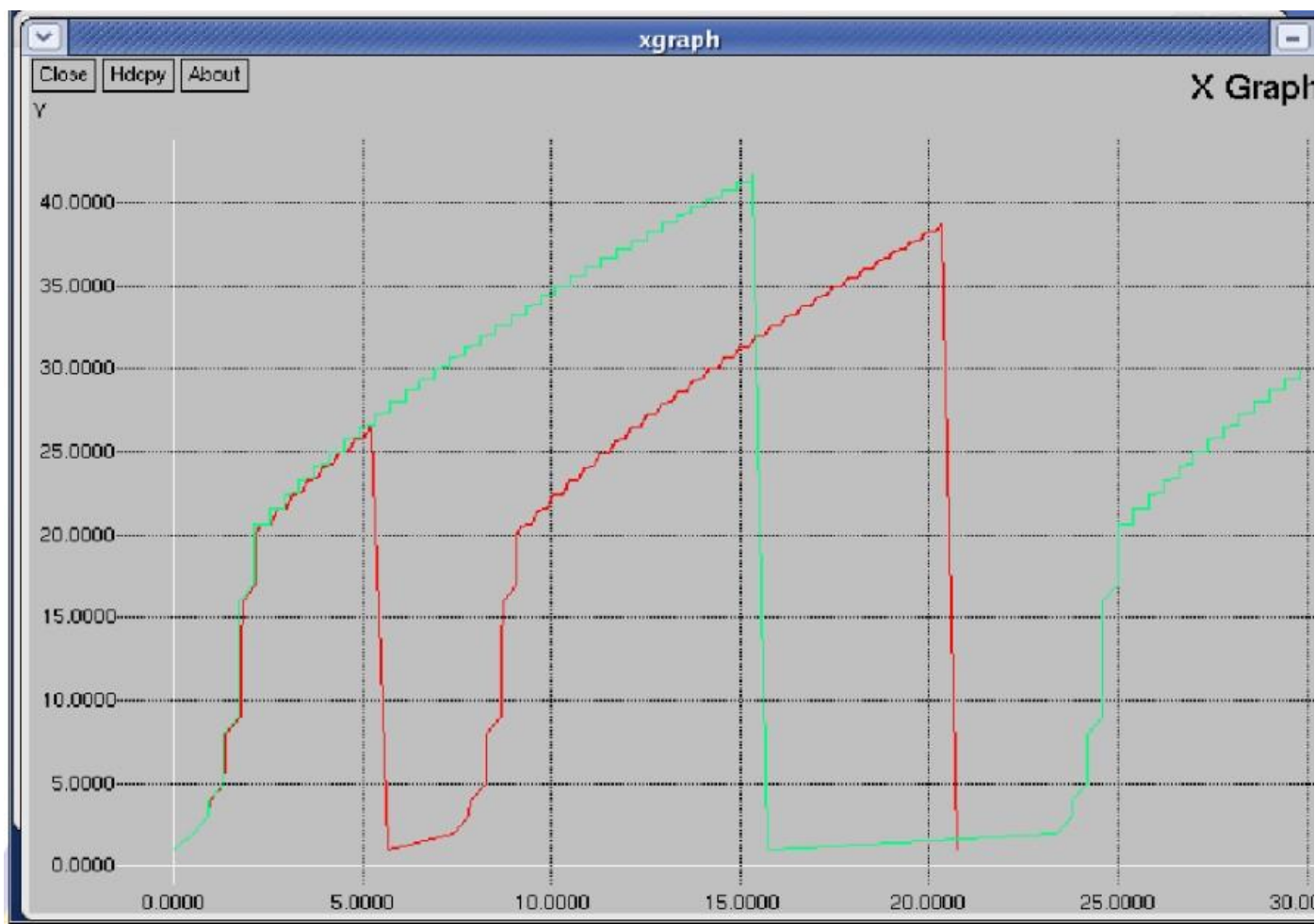
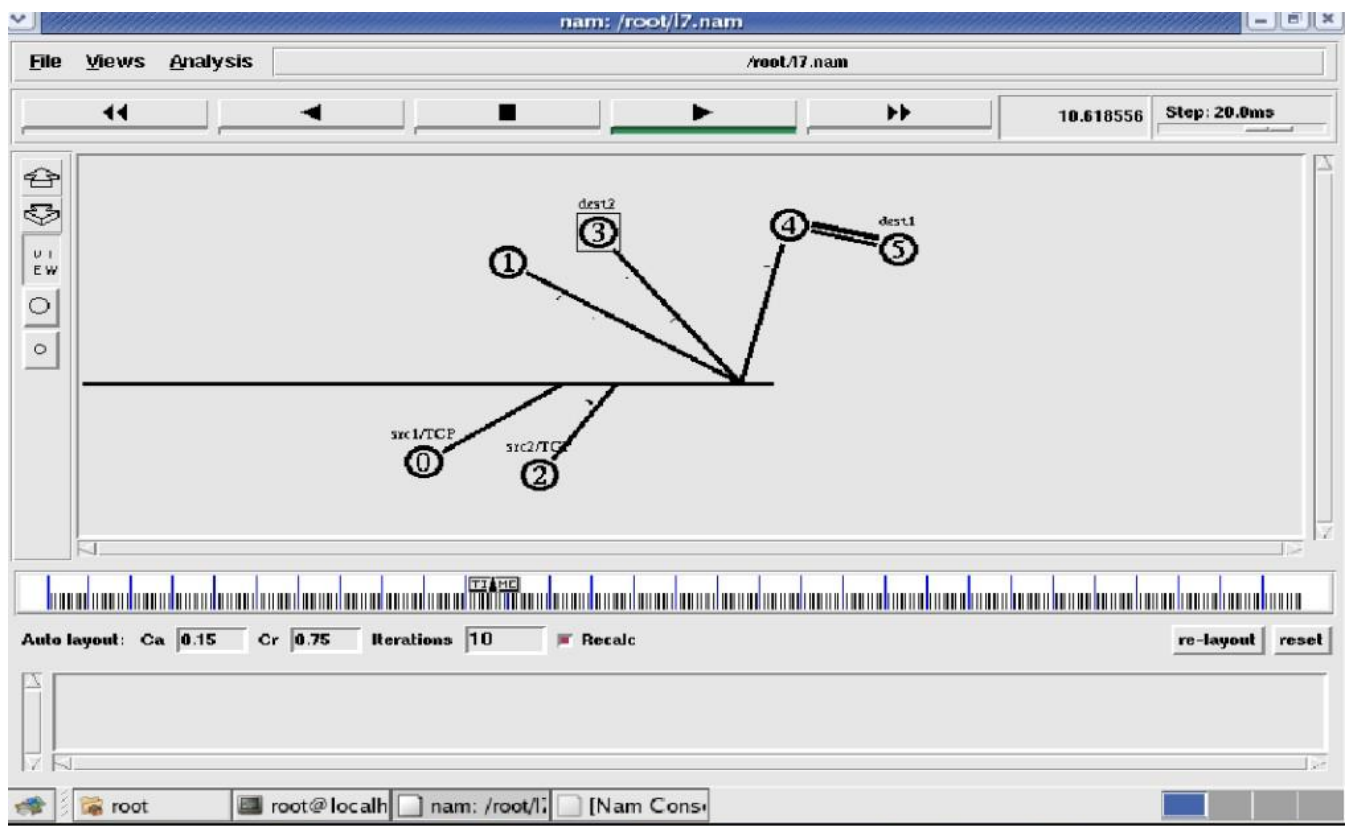
set file1 [open file1.tr w]
$tcp0 attach $file1
set file2 [open file2.tr w]
$tcp2 attach $file2
$tcp0 trace cwnd_
$tcp2 trace cwnd_
proc finish { } {
    global ns nf tf
    $ns flush-trace
    close $tf
    close $nf
    exec nam lab3.nam &
    exit 0
}
$ns at 0.1 "$ftp0 start"
$ns at 5 "$ftp0 stop"
$ns at 7 "$ftp0 start"
$ns at 0.2 "$ftp2 start"
$ns at 8 "$ftp2 stop"
$ns at 14 "$ftp0 stop"
$ns at 10 "$ftp2 start"
$ns at 15 "$ftp2 stop"
$ns at 16 "finish"
$ns run
lab3.awk
BEGIN {
}
{
    if($6=="cwnd_")
        printf("%f\t%f\t\n",$1,$7);
}
END {
}

```

Steps for execution:

- gedit lab3.tcl (to open the tcl program file, type the program and save.)
- gedit lab3.awk (to open the awk program file, type the program and save)
- ns lab3.tcl (to execute and simulate the tcl program)
- After simulation is completed run awk file to see the output ,
- awk -f lab3.awk file1.tr > a1
- awk -f lab3.awk file2.tr > a2
- xgraph a1 a2

Output Topology:



Experiment No: 4

Error Detecting code using CRC-CCITT

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

In Linux operating system Java libraries are preinstalled. It's very easy and convenient to compile and run Java programs in Linux environment. To compile and run Java Program is a two-step process:

1. Compile Java Program from Command Prompt
[root@host ~]# javac Filename.java

The Java compiler (Javac) compiles java program and generates a byte-code with the same file name and .class extension.

2. Run Java program from Command Prompt
[root@host ~]# java Filename

The java interpreter (Java) runs the byte-code and gives the respective output. It is important to note that in above command we have omitted the .class suffix of the byte-code (Filename.class).

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

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

$$\begin{array}{r}
 101 = 5 \\
 10011 / 1101101 \\
 \underline{10011} \\
 10000 \\
 \underline{00000} \\
 100001 \\
 \underline{10011} \\
 1110 = 14 = \text{remainder}
 \end{array}$$

With decimal calculations you can quickly check that 109 divided by 19 gives a quotient of 5 with 14 as the remainder. But what we also see in the scheme is that every bit extra to check only costs one binary comparison and in 50% of the cases one binary subtraction. You can easily increase the number of bits of the test data string—for example to 56 bits if we use our example value "*Lammert*"—and the result can be calculated with 56 binary comparisons and an average of 28 binary subtractions. This can be implemented in hardware directly with only very few transistors involved. Also software algorithms can be very efficient.

All of the CRC formulas you will encounter are simply checksum algorithms based on modulo-2 binary division where we ignore carry bits and in effect the subtraction will be equal to an *exclusive or* operation. Though some differences exist in the specifics across different CRC formulas, the basic mathematical process is always the same:

- The message bits are appended with c zero bits; this *augmented message* is the dividend
- A predetermined $c+1$ -bit binary sequence, called the *generator polynomial*, is the divisor
- The checksum is the c -bit remainder that results from the division operation

Table 1 lists some of the most commonly used generator polynomials for 16- and 32-bit CRCs. Remember that the width of the divisor is always one bit wider than the remainder. So, for example, you'd use a 17-bit generator polynomial whenever a 16-bit checksum is required.

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

International Standard CRC Polynomials

Program:

```
import java.io.*;
class CRC
{
    public static void main(String args[]) throws IOException
    {
        BufferedReader br=new BufferedReader(new
        InputStreamReader(System.in));
        int[ ] data;
        int[ ] div; int[ ] divisor; int[ ] rem;
        int[ ] crc;
        int data_bits, divisor_bits, tot_length;
        System.out.println("Enter number of data bits : ");
        data_bits=Integer.parseInt(br.readLine());
        data=new int[data_bits];
        System.out.println("Enter data bits : ");
        for(int i=0; i<data_bits; i++)
            data[i]=Integer.parseInt(br.readLine());
        System.out.println("Enter number of bits in 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());
        tot_length=data_bits+divisor_bits-1;
        div=new int[tot_length];
        rem=new int[tot_length];
        crc=new int[tot_length];
        /* _____CRC GENERATION_____ */
        for(int i=0;i<data.length;i++)
```

```
div[i]=data[i];
System.out.print("Dividend (after appending 0's) are : ");
for(int i=0; i< div.length; i++)
System.out.print(div[i]);
System.out.println();
for(int j=0; j<div.length; j++)
{
rem[j] = div[j];
}
rem=divide(div,divisor,rem);
for(int i=0;i<div.length;i++) //append dividend and 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());
for(int j=0; j<crc.length; j++)
{
rem[j] = crc[j];
}
rem=divide(crc,divisor,rem);
for(int i=0; i< rem.length; i++)
{
if(rem[i]!=0)
{
System.out.println("Error");
break;
}
}
if(i==rem.length-1)
System.out.println("No Error");
}

System.out.println("THANK YOU .....");
}
static int[] divide(int div[],int divisor[], int rem[])
{
int cur=0;
while(true)
{
for(int i=0;i<divisor.length;i++)
rem[cur+i]=(rem[cur+i]^divisor[i]);
```

```
while(rem[cur]==0 && cur!=rem.length-1)
cur++;
if((rem.length-cur)<divisor.length)
break;
}
return rem;
}
}
```

Output:



```
File Edit View Terminal Help
[root@localhost ~]# javac Crc.java
[root@localhost ~]# java Crc
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
Error
THANK YOU.... :)
[root@localhost ~]#
```

Experiment 5:

Experiment No: 6**Shortest Path- BellmanFord**

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

Distance Vector Algorithm is a decentralized routing algorithm that requires that each router simply inform its neighbors of its routing table. For each network path, the receiving routers pick the neighbor advertising the lowest cost, then add this entry into its routing table for re-advertisement. To find the shortest path, Distance Vector Algorithm is based on one of two basic algorithms: the Bellman-Ford and the Dijkstra algorithms.

Routers that use this algorithm have to maintain the distance tables (which is a one- dimension array -- "a vector"), which tell the distances and shortest path to sending packets to each node in the network. The information in the distance table is always up to date by exchanging information with the neighboring nodes. The number of data in the table equals to that of all nodes in networks (excluded itself). The columns of table represent the directly attached neighbors whereas the rows represent all destinations in the network. Each data contains the path for sending packets to each destination in the network and distance/or time to transmit on that path (we call this as "cost"). The measurements in this algorithm are the number of hops, latency, the number of outgoing packets, etc.

The Bellman-Ford algorithm is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph. It is slower than Dijkstra's algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers. Negative edge weights are found in various applications of graphs, hence the usefulness of this algorithm. If a graph contains a "negative cycle" (i.e. a cycle whose edges sum to a negative value) that is reachable from the source, then there is no cheapest path: any path that has a point on the negative cycle can be made cheaper by one more walk around the negative cycle. In such a case, the Bellman-Ford algorithm can detect negative cycles and report their existence

Source code:

```
import java.util.Scanner;

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

    public BellmanFord(int 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;
        }
    }
}
```

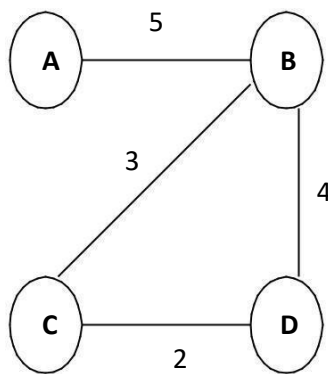
```

    }
    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]);
    }
}

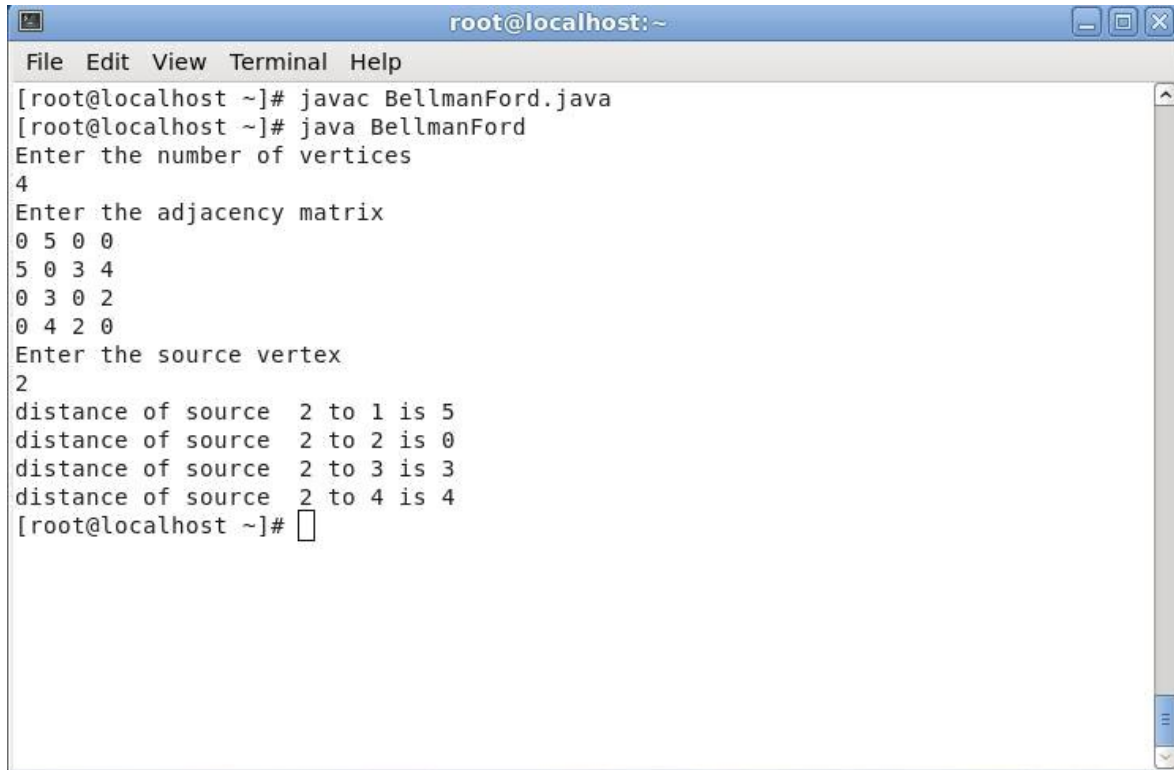
public static void main(String[ ] args)
{
    int num_ver = 0;
    int source;
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter the number of
    vertices"); num_ver = scanner.nextInt();
    int A[][] = new int[num_ver + 1][num_ver + 1];
    System.out.println("Enter the adjacency 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;
            continue;
        }
        if (A[sn][dn] == 0)
        {
            A[sn][dn] = MAX_VALUE;
        }
    }
}
System.out.println("Enter the source vertex");
source = scanner.nextInt();
BellmanFord b = new BellmanFord
(num_ver); b.BellmanFordEvaluation(source,
A); scanner.close();
}
}
```

Input graph:**Output:**

[root@localhost ~]# vi BellmanFord.java



```
root@localhost: ~  
File Edit View Terminal Help  
[root@localhost ~]# javac BellmanFord.java  
[root@localhost ~]# java BellmanFord  
Enter the number of vertices  
4  
Enter the adjacency matrix  
0 5 0 0  
5 0 3 4  
0 3 0 2  
0 4 2 0  
Enter the source vertex  
2  
distance of source 2 to 1 is 5  
distance of source 2 to 2 is 0  
distance of source 2 to 3 is 3  
distance of source 2 to 4 is 4  
[root@localhost ~]#
```

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. Implement the above program using as message queues or FIFOs as IPC channels.

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

Source Code:

TCP Client

```
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.EOFException;
import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.net.Socket;
import java.util.Scanner;

class Client
{
    public static void main(String args[])throws Exception
    {
        String address = "";
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter Server Address: ");
        address=sc.nextLine();
        //create the socket on port 5000
        Socket s=new Socket(address,5000);
        DataInputStream din=new DataInputStream(s.getInputStream());
        DataOutputStream dout=new DataOutputStream(s.getOutputStream());
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

        System.out.println("Send Get to start...");
        String str="",filename="";
        try
        {
            while(!str.equals("start"))
            str=br.readLine();
            dout.writeUTF(str);
            dout.flush();
            filename=din.readUTF();
            System.out.println("Receiving file: "+filename);
            filename="client"+filename;
            System.out.println("Saving as file: "+filename);
        }
    }
}
```

```
long sz=Long.parseLong(din.readUTF()); System.out.println
("File Size: "+(sz/(1024*1024))+" MB");
byte b[]=new byte [1024];
System.out.println("Receiving file..");
FileOutputStream fos=new FileOutputStream(new File(filename),true);
long bytesRead;
do
{
    bytesRead = din.read(b, 0, b.length);
    fos.write(b,0,b.length);
}while(!(bytesRead<1024));
System.out.println("Completed");
fos.close();
dout.close();
s.close();
}
catch(EOFException e)
{
    //do nothing
}
```

```
}  
}
```

TCP Server

```
import java.io.DataInputStream;  
import java.io.DataOutputStream;  
import java.io.File;  
import java.io.FileInputStream;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.util.Scanner;  
class Server  
{  
    public static void main(String args[])throws Exception  
    {  
        String filename;  
        System.out.println("Enter File Name: ");  
        Scanner sc=new Scanner(System.in);  
        filename=sc.nextLine();  
        sc.close();  
        while(true)  
        {  
            //create server socket on port 5000  
            ServerSocket ss=new ServerSocket(5000);  
            System.out.println ("Waiting for request");  
            Socket s=ss.accept();  
            System.out.println ("Connected With "+s.getInetAddress().toString());  
            DataInputStream din=new DataInputStream(s.getInputStream());  
            DataOutputStream dout=new DataOutputStream(s.getOutputStream());  
            try  
            {  
                String str="";  
                str=din.readUTF();  
                System.out.println("SendGet. ..Ok");  
                if(!str.equals("stop")){  
                    System.out.println("Sending File: "+filename);  
                    dout.writeUTF(filename);  
                    dout.flush();  
                    File f=new File(filename);  
                    FileInputStream fin=new FileInputStream(f);  
                    long sz=(int) f.length();  
                    byte b[]=new byte [1024];  
                    int read;  
                    dout.writeUTF(Long.toString(sz));  
                    dout.flush();  
                    System.out.println ("Size: "+sz);
```


```
        System.out.println ("Buf size:
        "+ss.getReceiveBufferSize()); while((read = fin.read(b)) !=
        -1) {
            dout.write(b, 0, read);
            dout.flush();
        }
        fin.close();
        System.out.println("..ok");
        dout.flush();
    }
    dout.writeUTF("stop");
    System.out.println("Send Complete");
    dout.flush();
}
catch(Exception e)
{
    e.printStackTrace();
    System.out.println("An error occured");
}
    }
    din.close();
    s.close();
    ss.close();
}
}
```

Note: Create two different files Client.java and Server.java. Follow the steps given:

1. Open a terminal run the server program and provide the filename to send
2. Open one more terminal run the client program and provide the IP address of the server. We can give localhost address "127.0.0.1" as it is running on same machine or give the IP address of the machine.
3. Send any start bit to start sending file.
4. Refer https://www.tutorialspoint.com/java/java_networking.htm for all the parameters, methods description in socket communication.

Output:

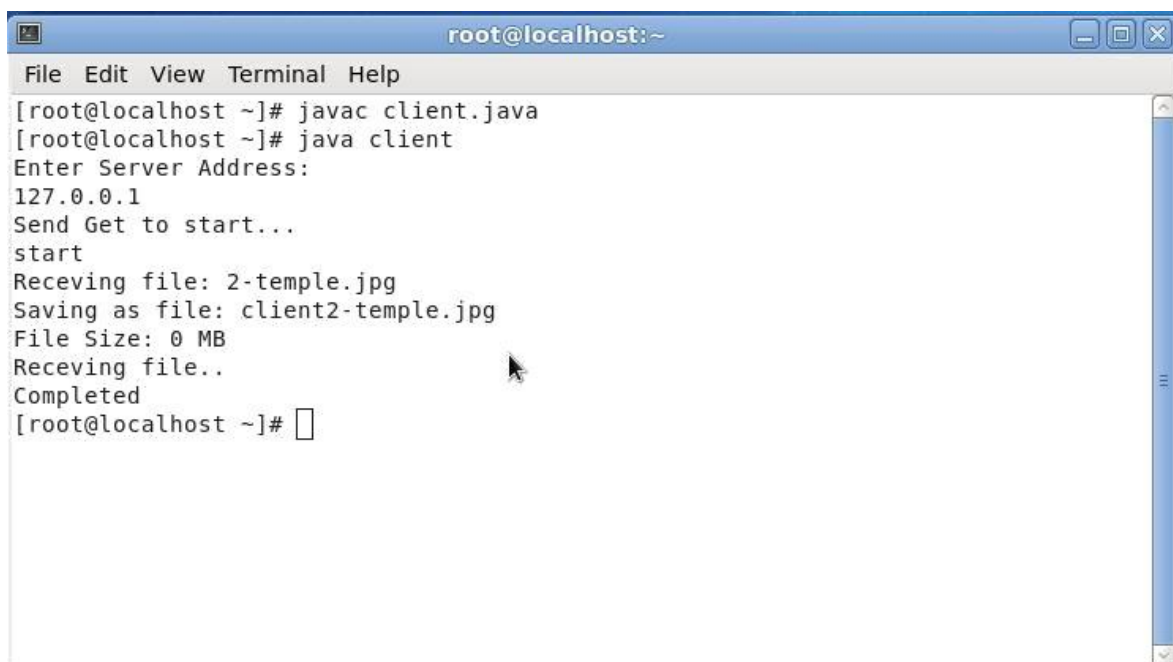
At server side:

A screenshot of a terminal window titled 'root@localhost:~'. The window has a menu bar with 'File', 'Edit', 'View', 'Terminal', and 'Help'. The terminal shows the following commands and output:

```
[root@localhost ~]# javac server.java
[root@localhost ~]# java server
Enter File Name:
2-temple.jpg
Waiting for request
Connected With /127.0.0.1
SendGet...Ok
Sending File: 2-temple.jpg
Size: 19868
Buf size: 43690
..ok
Send Complete
Waiting for request

```

At client side:

A screenshot of a terminal window titled 'root@localhost:~'. The window has a menu bar with 'File', 'Edit', 'View', 'Terminal', and 'Help'. The terminal shows the following commands and output:

```
[root@localhost ~]# javac client.java
[root@localhost ~]# java client
Enter Server Address:
127.0.0.1
Send Get to start...
start
Receiving file: 2-temple.jpg
Saving as file: client2-temple.jpg
File Size: 0 MB
Receiving file..
Completed
[root@localhost ~]# 
```

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

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

Source Code:**UDP Client**

```
import java.io.*;
import java.net.*;
public class UDPC
{
    public static void main(String[] args)
    {
        DatagramSocket skt;
        try
        {
            skt=new DatagramSocket();
            String msg= "text message ";
            byte[] b = msg.getBytes();
            InetAddress host=InetAddress.getByName("127.0.0.1");
            int serverSocket=6788;
            DatagramPacket request =new DatagramPacket
                (b,b.length,host,serverSocket); skt.send(request);
            byte[] buffer =new byte[1000];
            DatagramPacket reply= new
                DatagramPacket(buffer,buffer.length); skt.receive(reply);
            System.out.println("client received:" +new
                String(reply.getData())); skt.close();
        }
        catch(Exception ex)
        {
        }
    }
}
```

UDP Server

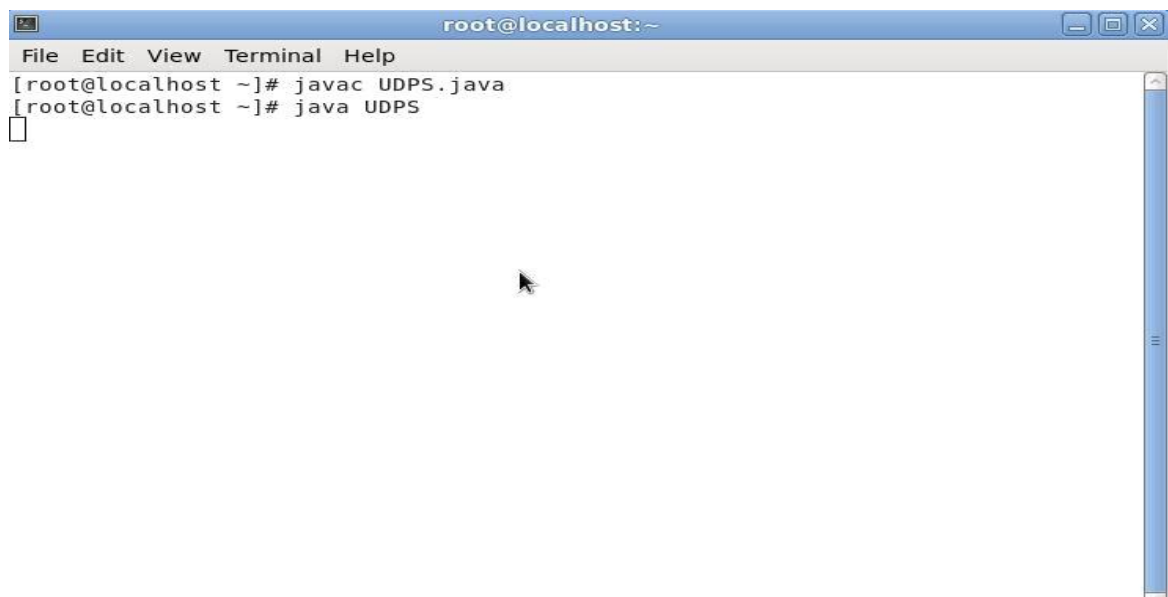
```
import java.io.*;
import java.net.*;
public class UDPS
{
    public static void main(String[] args)
    {
        DatagramSocket skt=null;
```

```
try
{
    skt=new DatagramSocket(6788);
    byte[] buffer = new byte[1000];
    while(true)
    {
        DatagramPacket request = new
            DatagramPacket(buffer,buffer.length); skt.receive(request);
        String[] message = (new String(request.getData())).split(" ");
        byte[] sendMsg= (message[1]+ " server processed").getBytes();
        DatagramPacket      reply      =      new
            DatagramPacket(sendMsg,sendMsg.length,request.getAddress
            (),request.getPort());
        skt.send(reply);
    }
}
catch(Exception ex)
{
}
}
```

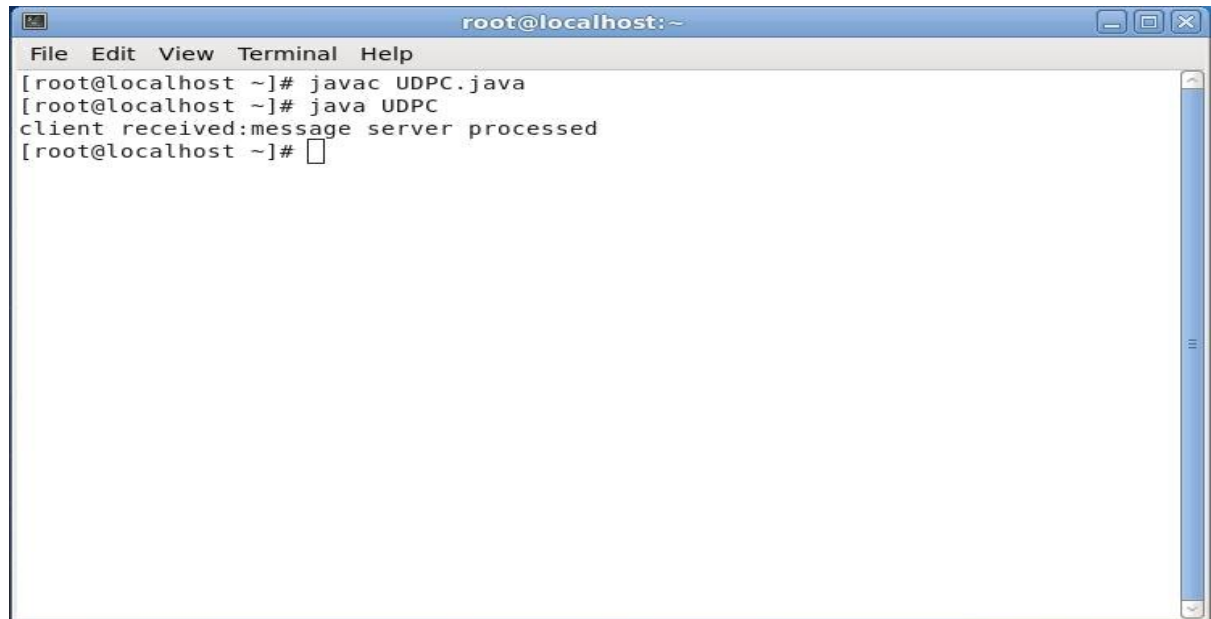
Note: Create two different files UDPC.java and UDPS.java. Follow the following steps:

1. Open a terminal run the server program.
2. Open one more terminal run the client program, the sent message will be received.

At Server side:



```
root@localhost:~
File Edit View Terminal Help
[root@localhost ~]# javac UDPS.java
[root@localhost ~]# java UDPS
```


At Client side:

```
root@localhost:~  
File Edit View Terminal Help  
[root@localhost ~]# javac UDPC.java  
[root@localhost ~]# java UDPC  
client received:message server processed  
[root@localhost ~]#
```

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

RSA is an example of public key cryptography. It was developed by Rivest, Shamir and Adelman. The RSA algorithm can be used for both public key encryption and digital signatures. Its security is based on the difficulty of factoring large integers.

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

Key Generation Algorithm

3. Generate two large random primes, p and q , of approximately equal size such that their product $n = p \cdot q$
4. Compute $n = p \cdot q$ and Euler's totient function (ϕ) $\phi(n) = (p-1)(q-1)$.
5. Choose an integer e , $1 < e < \phi$, such that $\gcd(e, \phi) = 1$.
6. Compute the secret exponent d , $1 < d < \phi$, such that $e \cdot d \equiv 1 \pmod{\phi}$.
7. The public key is (e, n) and the private key is (d, n) . The values of p , q , and ϕ should also be kept secret.

Encryption

Sender A does the following:-

1. Using the public key (e,n)
2. Represents the plaintext message as a positive integer M
3. Computes the cipher text $C = M^e \bmod n$.
4. Sends the cipher text C to B (Receiver).

Decryption

Recipient B does the following:-

1. Uses his private key (d, n) to compute $M = C^d \bmod n$.
2. Extracts the plaintext from the integer representative m.

Source Code:

RSA Key Generation

```
import java.util.*;
import java.math.BigInteger;
import java.lang.*;

class RSAkeygen
{
    public static void main(String[] args)
    {
        Random rand1=new Random(System.currentTimeMillis());
        Random rand2=new Random(System.currentTimeMillis()*10);

        int pubkey=Integer.parseInt(args[0]);

        BigInteger bigB_p=BigInteger.probablePrime(32, rand1);
        BigInteger bigB_q=BigInteger.probablePrime(32, rand2);

        BigInteger bigB_n=bigB_p.multiply(bigB_q);

        BigInteger bigB_p_1=bigB_p.subtract(new BigInteger("1"));
        BigInteger bigB_q_1=bigB_q.subtract(new BigInteger("1"));

        BigInteger bigB_p_1_q_1=bigB_p_1.multiply(bigB_q_1);

        while(true)
        {
            BigInteger BigB_GCD=bigB_p_1_q_1.gcd(new BigInteger(""+pubkey));
            if(BigB_GCD.equals(BigInteger.ONE))
            {
                break;
            }
            pubkey++;
        }
        BigInteger bigB_pubkey=new BigInteger(""+pubkey);
```

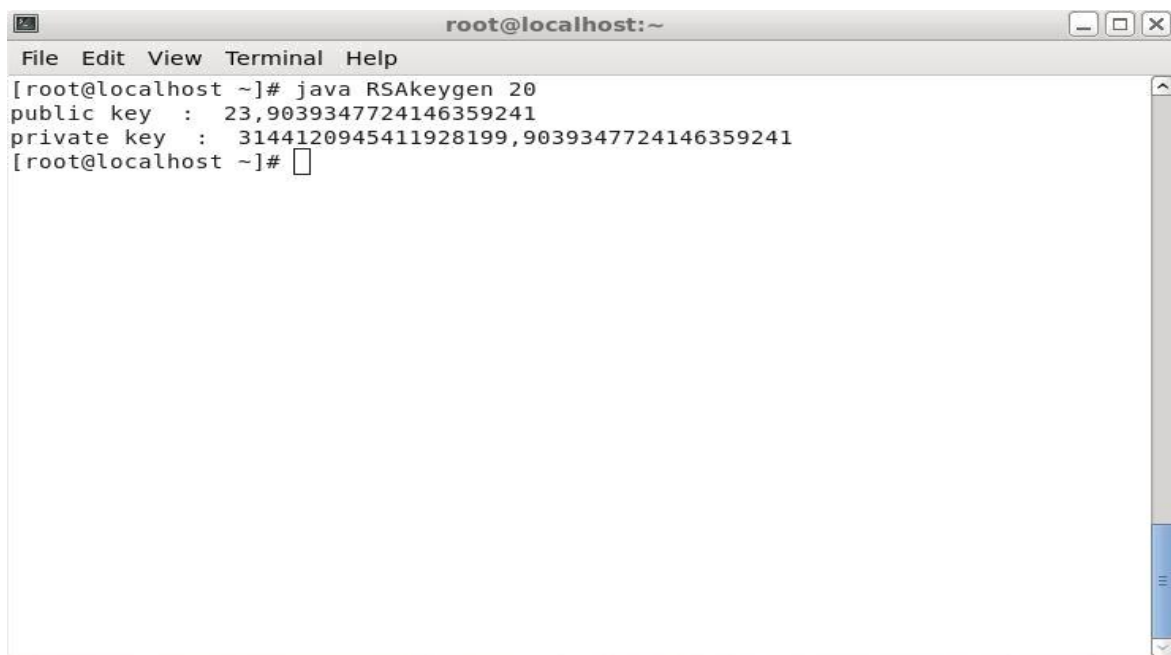
```
        BigInteger bigB_prvkey=bigB_pubkey.modInverse(bigB_p_1_q_1);
        System.out.println("public key : "+bigB_pubkey+","+bigB_n);
        System.out.println("private key : "+bigB_prvkey+","+bigB_n);
    }
}
```

RSA Encryption and Decryption

```
import java.math.BigInteger;
import java.util.*;
class RSAEncDec
{
    public static void main(String[] args)
    {
        BigInteger bigB_pubkey = new BigInteger(args[0]);
        BigInteger bigB_prvkey = new BigInteger(args[1]);
        BigInteger bigB_n = new BigInteger(args[2]); int
        asciiVal=Integer.parseInt(args[3]);
        BigInteger bigB_val=new BigInteger(""+asciiVal);
        BigInteger bigB_cipherVal=bigB_val.modPow(bigB_pubkey, bigB_n);
        System.out.println("Cipher text: " + bigB_cipherVal);
        BigInteger bigB_plainVal=bigB_cipherVal.modPow(bigB_prvkey, bigB_n);
        int plainVal=bigB_plainVal.intValue();
        System.out.println("Plain text:" + plainVal);
    }
}
```

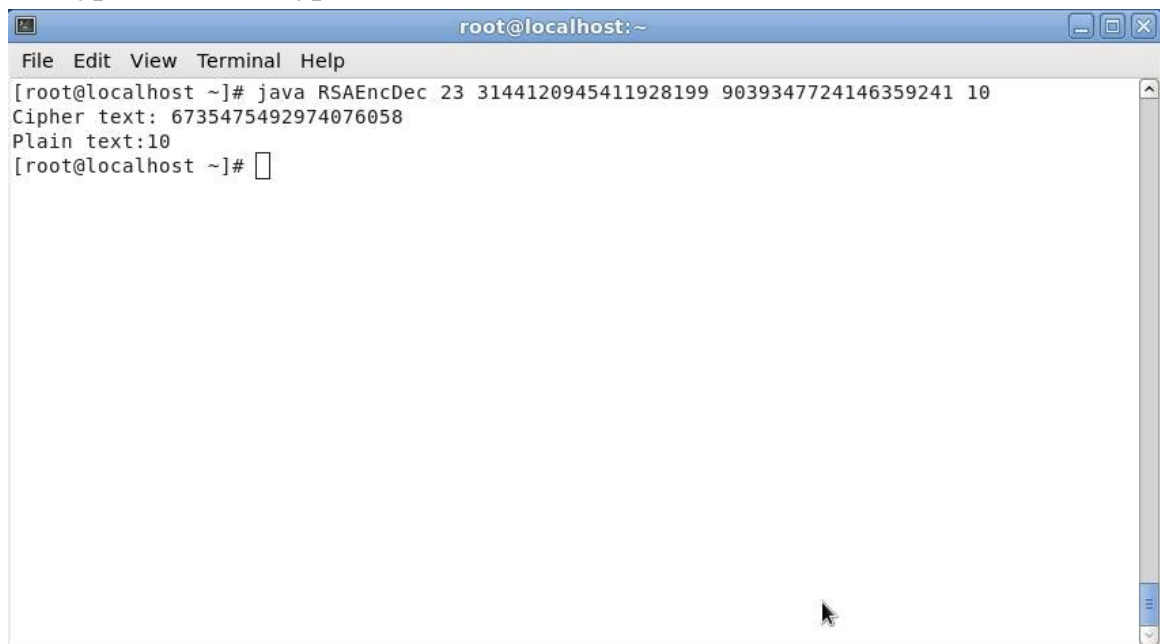
Output:

Key Generation



```
root@localhost:~
File Edit View Terminal Help
[root@localhost ~]# java RSAkeygen 20
public key : 23,9039347724146359241
private key : 3144120945411928199,9039347724146359241
[root@localhost ~]#
```

Encryption and Decryption

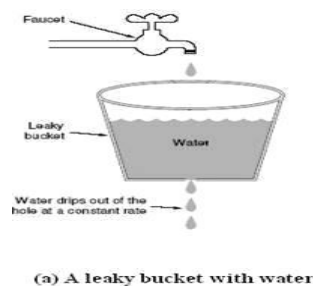


The screenshot shows a Java application window titled "root@localhost:~". The window contains a menu bar with "File", "Edit", "View", "Terminal", and "Help". The main text area displays the following output:

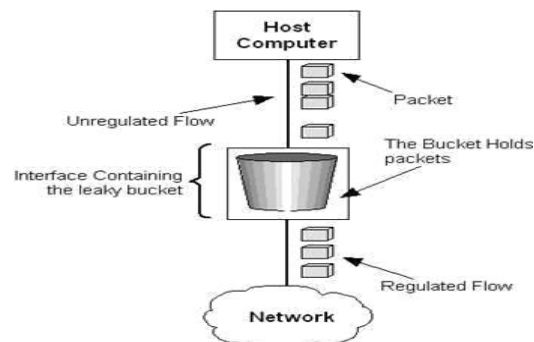
```
[root@localhost ~]# java RSAEncDec 23 3144120945411928199 9039347724146359241 10
Cipher text: 6735475492974076058
Plain text:10
[root@localhost ~]#
```

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

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



(a) A leaky bucket with water



(b) A leaky bucket with packets

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

Source Code:

```
import java.io.*;
import java.util.*;
class Queue
{
    int q[],f=0,r=0,size;
    void insert(int n)
    {
        Scanner in = new Scanner(System.in);
        q=new int[10];
        for(int i=0;i<n;i++)
        {
            System.out.print("\nEnter " + i + " element: ");
            int ele=in.nextInt();
            if(r+1>10)
            {
```

```

        System.out.println("\nQueue is full \nLost Packet:
        "+ele); break;
    }
    else
    {
        r++;
        q[i]=ele;
    }
}

}

void delete()
{
    Scanner in = new Scanner(System.in);
    Thread t=new Thread();
    if(r==0)
        System.out.print("\nQueue empty ");
    else
    {
        for(int i=f;i<r;i++)
        {
            try
            {
                t.sleep(1000);
            }

            catch(Exception e){ }
            System.out.print("\nLeaked Packet: "+q[i]);
            f++;
        }
        System.out.println();
    }
}

}

class Leaky extends Thread
{
    public static void main(String ar[]) throws Exception
    {
        Queue q=new Queue();
        Scanner src=new Scanner(System.in);
        System.out.println("\nEnter the packets to be sent:");
        int size=src.nextInt();
        q.insert(size);
        q.delete();
    }
}

```

Output:

```
File Edit View Terminal Help
```

```
Enter the packets to be sent:  
12
```

```
Enter 0 element: 2
```

```
Enter 1 element: 3
```

```
Enter 2 element: 5
```

```
Enter 3 element: 6
```

```
Enter 4 element: 8
```

```
Enter 5 element: 9
```

```
Enter 6 element: 4
```

```
Enter 7 element: 5
```

```
Enter 8 element: 6
```

```
Enter 9 element: 2
```

```
Enter 10 element: 3
```

```
Queue is full  
Lost Packet: 3
```

```
Leaked Packet: 2  
Leaked Packet: 3  
Leaked Packet: 5  
Leaked Packet: 6  
Leaked Packet: 8  
Leaked Packet: 9  
Leaked Packet: 4
```

REFERENCES

1. <https://www.isi.edu/nsnam/ns/>
2. <http://aknetworks.webs.com/e-books>
3. **Communication Networks: Fundamental Concepts and Key Architectures** - Alberto Leon, Garcia and Indra Widjaja, 4th Edition, Tata McGraw- Hill, reprint-2012.
4. **Data and Computer Communication**, William Stallings, 8th Edition, Pearson Education, 2009.
5. **Computer Networks: A Systems Approach** - Larry L. Peterson and Bruce S. David, 4th Edition, Elsevier, 2009.
6. **Introduction to Data Communications and Networking** – Wayne Tomasi, Pearson Education, 2009.
7. **Communication Networks – Fundamental Concepts and Key architectures** – Alberto Leon- Garcia and Indra Widjaja:, 2rd Edition, Tata McGraw-Hill, 2009
8. **Computer and Communication Networks** – Nader F. Mir:, Pearson Education, 2012