Department of Information Science & Engineering

LAB MANUAL

(Scheme 2018)

18CSL57 COMPUTER NETWORK LABORATORY

V Semester CSE

Name : _____

U S N : _____

Batch : _____ **Section :** _____

‘Instructions to the Candidates’

1. Students should come with thorough preparation for the experiment to be conducted.

2. Students will not be permitted to attend the laboratory unless they bring the practical record fully completed in all respects pertaining to the experiment conducted in the previous class.

3. Practical record should be neatly maintained.

4. They should obtain the signature of the staff-in-charge in the observation book after completing each experiment.

5. Theory regarding each experiment should be written in the practical record before procedure in your own words.

6. Ask lab technician for assistance if you have any problem.

7. Save your class work, assignments in system .

8. Do not download or install software without the assistance of the laboratory technician.

9. Do not alter the configuration of the system.

10. Turnoff the systems after use.

TABLE OF CONTENTS

Part-B

Part-B

Experiment No: 1 **Date:**

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

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

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

```
                    1 0 1 = 5
                 -------------
1 0 0 1 1 / 1 1 0 1 1 0 1
            1 0 0 1 1 | |
            --------- | |
              1 0 0 0 0 |
              0 0 0 0 0 |
              --------- |
              1 0 0 0 0 1
                1 0 0 1 1
                ---------
                  1 1 1 0 = 14 = remainder
```

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

young. It might only look a little bit strange. Also notations differ between countries, but the method is similar.

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

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

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

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

Table 1: International Standard CRC Polynomials

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

Error detection with CRC

Consider a message represented by the polynomial M(x)

Consider a *generating polynomial* G(x)

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

Note this G(x) is prime.

Steps:

1. Multiply M(x) by highest power in G(x). i.e. Add So much zeros to M(x).
2. Divide the result by G(x). The remainder = C(x).

   Special case: This won't work if bitstring =all zeros. We don't allow such an M(x).But M(x) bitstring = 1 will work, for example. Can divide 1101 into 1000.
3. If: x div y gives remainder c

   that means: x = n y + c

   Hence (x-c) = n y

   (x-c) div y gives remainder 0

   Here (x-c) = (x+c)

   Hence (x+c) div y gives remainder 0
4. Transmit: T(x) = M(x) + C(x)
5. Receiver end: Receive T(x). Divide by G(x), should have remainder 0.

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

Some CRC polynomials that are actually used

Some CRC polynomials

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

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

Source Code:

```
import java.util.*;
class crc
{       void div(int a[],int k)
        {       int gp[]={1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,1};
                int count=0;
                for(int i=0;i<k;i++)
                {
                        if(a[i]==gp[0])
                        {
                                for(int j=i;j<17+i;j++)
                                {
                                        a[j]=a[j]^gp[count++];
                                }
                                count=0;
                        }
                }
        }
public static void main(String args[])
{
        int a[]=new int[100];
        int b[]=new int[100];
        int len,k;
        crc ob=new crc();
        System.out.println("Enter the length of Data Frame:");
        Scanner sc=new Scanner(System.in);
        len=sc.nextInt();
        int flag=0;
        System.out.println("Enter the Message:");
        for(int i=0;i<len;i++)
        {       a[i]=sc.nextInt();
        }
        for(int i=0;i<16;i++)
        {       a[len++]=0;
        }
        k=len-16;
        for(int i=0;i<len;i++)
        {       b[i]=a[i];
        }
        ob.div(a,k);
                for(int i=0;i<len;i++)
                a[i]=a[i]^b[i];
        System.out.println("Data to be transmitted: ");
        for(int i=0;i<len;i++)
```

```
{
        System.out.print(a[i]+" ");
}
System.out.println();
System.out.println("Enter the Reveived Data: ");
for(int i=0;i<len;i++)
{
        a[i]=sc.nextInt();
}
ob.div(a, k);

for(int i=0;i<len;i++)
{
        if(a[i]!=0)
        {
                System.out.println("ERROR in Received data");
                return;
        }
        System.out.println("no error");
}

        }
}
```

Output:

Enter the length of Data Frame: 4

Enter the Message: 1 0 1 1

Data to be transmitted: 1 0 1 1 1 0 1 1 0 0 0 1 0 1 1 0 1 0 1 1

Enter the Received Data: 1 0 1 1 1 0 1 1 0 0 0 0 0 1 1 0 1 0 1 1

ERROR in Received Data

****

Experiment No: 2 **Date:**

Bellman-ford Algorithm

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

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

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

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

Implementation Algorithm:

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

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

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

Source Code:

```
import java.util.Scanner;
public class BellmanFord
{       private int D[];
        private int num_ver;
        public static final int MAX_VALUE = 999;
        public BellmanFord(int n)
        {       this.n=n;
                D = new int[n+1];
        }
        public void shortest(int s,int A[][])
        {       for (int i=1;i<=n;i++)
                {       D[i]=MAX_VALUE;
                }       D[s] = 0;
                for(int k=1;k<=n-1;k++)
                {       for(int i=1;i<=n;i++)
                        {       for(int j=1;j<=n;j++)
                                {       if(A[i][j]!=MAX_VALUE)
                                        {       if(D[j]>D[i]+A[i][j])
                                                D[j]=D[i]+A[i][j];
                                        }
                                }
                        }
                }
                for(int i=1;i<=n;i++)
                        {       for(int j=1;j<=n;j++)
                                {       if(A[i][j]!=MAX_VALUE)
                                        {       if(D[j]>D[i]+A[i][j])
                                                {
System.out.println("The Graph contains negative egde cycle");
                                                return;
                                        }       }
                                        }
                        }
for(int i=1;i<=n;i++)
{
System.out.println("Distance of source " + s + " to "+ i + " is " + D[i]);
}
}
public static void main(String[ ] args)
{       int n=0,s;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of vertices");
        n = sc.nextInt();
```

```
        int A[][] = new int[n+1][n+1];
        System.out.println("Enter the Weighted Matrix");
        for(int i=1;i<=n;i++)
        {       for(int j=1;j<=n;j++)
                {       A[i][j]=sc.nextInt();
                        if(i==j)
                        {       A[i][j]=0;
                                continue;
                        }
                        if(A[i][j]==0)
                        {       A[i][j]=MAX_VALUE;
                        }
                }
        }
        System.out.println("Enter the source vertex");
        s=sc.nextInt();
        BellmanFord b = new BellmanFord(n);
        b.shortest(s,A);
        sc.close();
}
}
```

Output:

Enter the number of vertices
4
Enter the Weighted Matrix
0 5 0 0
5 0 3 4
0 3 0 2
0 4 2 0
Enter the source vertex
2
Distance of source 2 to 1 is 5
Distance of source 2 to 2 is 0
Distance of source 2 to 3 is 3
Distance of source 2 to 4 is 4

Experiment No: 3 **Date:**

Client-server using TCP/IP sockets

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

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

Source Code:

TCP Server

The server program has three responsibilities which must be fulfilled in the code. First job is to read the file name coming from client. For this, it uses input stream. Second one is to open the file, using some input stream, and read the contents. Third one is, as the reading is going on, to send the contents each line separately.

```
import java.net.*;
import java.io.*;
public class ContentsServer
{       public static void main(String args[]) throws Exception
        { // establishing the connection with the server
        ServerSocket sersock = new ServerSocket(4000);
        System.out.println("Server ready for connection");
        Socket sock = sersock.accept();     // binding with port: 4000
        System.out.println("Connection is successful and waiting for chatting");
        // reading the file name from client
        InputStream istream = sock.getInputStream( );
        BufferedReader fileRead =new BufferedReader(new InputStreamReader(istream));
        String fname = fileRead.readLine( );
        // reading file contents
        BufferedReader contentRead = new BufferedReader(new FileReader(fname) );
```

```
        // keeping output stream ready to send the contents
        OutputStream ostream = sock.getOutputStream( );
        PrintWriter pwrite = new PrintWriter(ostream, true);
        String str;
        // reading line-by-line from file
        while((str = contentRead.readLine()) != null)
         {  pwrite.println(str);       // sending each line to client
         }
        sock.close();
        sersock.close();      // closing network sockets
        pwrite.close();
        fileRead.close();
        contentRead.close();
        }
}
```

TCP Client:

To read filename as input from keyboard, remember, this file should exist on server. For this, it uses input stream. The file read from keyboard should be sent to the server. For this client uses output stream. The file contents sent by the server, the client should receive and print on the console.

BufferedReader:

The System.in is a byte stream and cannot be chained to BufferedReader as BufferedReader is a character stream. The byte stream System.in is to be converted (wrapped) into a character stream and then passed to BufferedReader constructor. To take input from the keyboard, a BufferedReader object, keyRead, is created. To send the file name to the server, pwrite of PrintWriter and to receive the file contents from the server, socketRead of BufferedReader are created.

PrintWriter pwrite = new PrintWriter(ostream, true);

```
import java.net.*;
import java.io.*;
public class ContentsClient
{       public static void main( String args[ ] ) throws Exception
```

```
        {
        Socket sock = new Socket( "127.0.0.1", 4000);
         // reading the file name from keyboard. Uses input stream
        System.out.print("Enter the file name");
BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));
        String fname = keyRead.readLine();
          // sending the file name to server. Uses PrintWriter
        OutputStream ostream = sock.getOutputStream( );
        PrintWriter pwrite = new PrintWriter(ostream, true);
        pwrite.println(fname);
        // receiving the contents from server. Uses input stream
        InputStream istream = sock.getInputStream();
BufferedReader socketRead = new BufferedReader(new InputStreamReader(istream));
        String str;
        while((str = socketRead.readLine())  !=  null)          // reading line-by-line
        {       System.out.println(str);
        }        pwrite.close();
                socketRead.close();
                keyRead.close();
        }
}
```

Output:

At server side:

[root@localhost]# **javac ContentsServer.java**

[root@localhost]# **Java ContentsServer**

Server ready for connection

Connection is successful and waiting for chatting

At Client Side:

[root@localhost]# **javac ContentsClient.java**

[root@localhost]# **java ContentsClient**

Enter the file name

aa.txt

Welcome to Network Lab

Experiment No: 4 **Date:**

Client-Server Communication

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

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

Here, the connectionServer.java program creates a ServerSocket object bound to port 3456 and waits for an incoming client connection request. When a client contacts the server program, the accept( ) method is unblocked and returns a Socket object for the server to communicate with the particular client that contacted.

The server program then creates a PrintStream object through the output stream extracted from this socket and uses it to send a welcome message to the contacting client. The client program runs as follows: The client creates a Socket object to connect to the server running at the specified IP address or hostname and at the port number 3456. The client creates a BufferedReader object through the input stream extracted from this socket and waits for an incoming line of message from the other end. The readLine( ) method of the BufferedReader object blocks the client from proceeding further unless a line of message is received. The purpose of the flush( ) method of the PrintStream class is to write any buffered output bytes to the underlying output stream and then flush that stream to send out the bytes. Note that the server program in our example sends a welcome message to an incoming client request and then stops.

Source Code:

UDP Client

```
import java.net.*;
import java.io.*;
class connectionClient
{       public static void main(String[ ] args)
        {
        try
```

```
        {       InetAddress acceptorHost = InetAddress.getByName(args[0]);
                int serverPortNum = Integer.parseInt(args[1]);
                Socket clientSocket = new Socket(acceptorHost, serverPortNum);
BufferedReader br = new BufferedReader(new InputStreamReader(clientSocket.getInputStream( )));
                System.out.println(br.readLine( ));
                clientSocket.close();
        }
        catch(Exception e)
        { e.printStackTrace( );
        }
        }
}
```

UDP Server

```
import java.net.*;
import java.io.*;
class connectionServer
{       public static void main(String[ ] args)
        {
        try
        {               String message = args[0];
                        int serverPortNumber = Integer.parseInt(args[1]);
                ServerSocket connectionSocket = new ServerSocket(serverPortNumber);
                        Socket dataSocket = connectionSocket.accept();
                PrintStream socketOutput = new PrintStream(dataSocket.getOutputStream());
                        socketOutput.println(message);
                        System.out.println("sent response to client");
                        socketOutput.flush( );
                        dataSocket.close( );
                        connectionSocket.close( );
                        }
                catch(Exception e)
                { e.printStackTrace( );
                }
        }
```

}

Output:

Server Side

[root@localhost]# **Javac ConnectionServer.java**

[root@localhost]# **Java ConnectionServer "Welcome to Computer Network Lab" 3956**

Client Side:

[root@localhost]# **Javac ConnectionClient.java**

[root@localhost]# **Java ConnectionClient localhost 3956**

Welcome to Computer Network Lab

Experiment No: 5 **Date:**

RSA Algorithm to Encrypt and Decrypt the Data

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

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

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

A very simple example of RSA encryption

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

1. Select primes p = 11, q = 3.
2. n = pq = 11.3 = 33

   phi = (p-1)(q-1) = 10.2 = 20
3. Choose e=3

   Check gcd(e, p-1) = gcd(3, 10) = 1 (i.e. 3 and 10 have no common factors except 1), and check gcd(e, q-1) = gcd(3, 2) = 1

   therefore gcd(e, phi) = gcd(e, (p-1)(q-1)) = gcd(3, 20) = 1
4. Compute d such that ed ≡ 1 (mod phi)

   i.e. compute d = e^{-1} mod phi = 3^{-1} mod 20

   i.e. find a value for d such that phi divides (ed-1)

   i.e. find d such that 20 divides 3d-1.

   Simple testing (d = 1, 2, ...) gives d = 7

   Check: ed-1 = 3.7 - 1 = 20, which is divisible by phi.
5. Public key = (n, e) = (33, 3)

   Private key = (n, d) = (33, 7).

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

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

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

Hence the ciphertext c = 13.

To check decryption we compute

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

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

One way of calculating m' is as follows:-

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

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

$= (2197 \bmod 33).(2197 \bmod 33).(13 \bmod 33) \bmod 33$

$= 19.19.13 \bmod 33 = 4693 \bmod 33$

$= 7.$

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

m 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

c 0 1 8 27 31 26 18 13 17 3 10 11 12 19 5 9 4

m 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32

c 29 24 28 14 21 22 23 30 16 20 15 7 2 6 25 32

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

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

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

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

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

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

Remember that calculating m^e mod n is easy, but calculating the inverse c^-e mod n is very difficult, well, for large n's anyway. However, if we can factor n into its prime factors

p and q, the solution becomes easy again, even for large n's. Obviously, if we can get hold of the secret exponent d, the solution is easy, too.

Key Generation Algorithm

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

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

Encryption

Sender A does the following:-

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

Decryption

Recipient B does the following:-

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

Source Code:

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

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

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

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

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

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

	}
	nofelem=message.length();
	for(i=0;i<nofelem;i++)
	{
		encrypted[i]=1;
		for(j=0;j<e;j++)
		encrypted[i] =(encrypted[i]*nummes[i])%n;
	}
	System.out.println("\n Encrypted message\n");
	for(i=0;i<nofelem;i++)
	{
		System.out.print(encrypted[i]);
		System.out.print((char)(encrypted[i]+96));
	}
	for(i=0;i<nofelem;i++)
	{	decrypted[i]=1;
		for(j=0;j<d;j++)
```

```
                    decrypted[i]=(decrypted[i]*encrypted[i])%n;
                }

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

        }
```

#**********

****RESULT****

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

Experiment No: 6 **Date:**

Congestion Control Using Leaky Bucket Algorithm

Aim: *C Program for Congestion control using Leaky Bucket Algorithm*

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

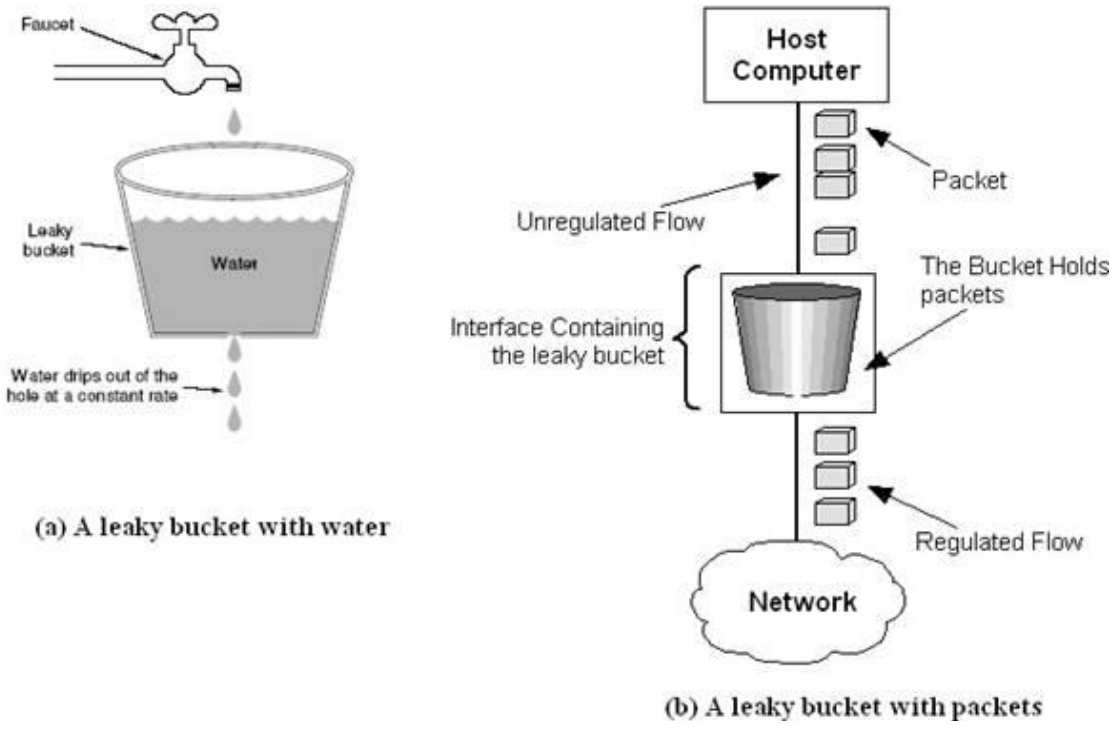

Figure 2.4 - The leaky bucket traffic shaping algorithm

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

Implementation Algorithm:

Steps:

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

Source Code:

```
import java.util.*;
public class leaky
{
	static int min(int x,int y)
	{
	if(x<y)
	return x;
	else
	return y;
	}
	public static void main(String[] args)
	{	int drop=0,mini,nsec,cap,count=0,i,process;
		int inp[]=new int[25];
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter The Bucket Size\n");
		cap= sc.nextInt();
		System.out.println("Enter The Operation Rate\n");
		process= sc.nextInt();
		System.out.println("Enter The No. Of Seconds You Want To Stimulate\n");
		nsec=sc.nextInt();
		for(i=0;i<nsec;i++)
		{	System.out.print("Enter The Size Of The Packet Entering At "+ i+1 + "
sec");
			inp[i] = sc.nextInt();
```

```
                }
                System.out.println("\nSecond | Packet Recieved | Packet Sent | Packet Left |
Packet Dropped|\n");
        System.out.println("-----------------------------------------------------------------\n");
                for(i=0;i<nsec;i++)
                {       count+=inp[i];
                        if(count>cap)
                        {               drop=count-cap;
                                count=cap;
                        }
                        System.out.print(i+1);
                        System.out.print("\t\t"+inp[i]);
                        mini=min(count,process);
                        System.out.print("\t\t"+mini);
                        count=count-mini;
                        System.out.print("\t\t"+count);
                        System.out.print("\t\t"+drop);
                        drop=0;
                        System.out.println();
                        }
                        for(;count!=0;i++)
                        {
                        if(count>cap)
                        {
                        drop=count-cap;
                        count=cap;
                        }
                        System.out.print(i+1);
                        System.out.print("\t\t0");
                        mini=min(count,process);
                        System.out.print("\t\t"+mini);
                        count=count-mini;
                        System.out.print("\t\t"+count);
                        System.out.print("\t\t"+drop);
                        System.out.println();
                        }
                        }
   }
```

Output:

```
Enter The Bucket Size
5
Enter The Operation Rate
2
Enter The No. Of Seconds You Want To Stimulate
3
Enter The Size Of The Packet Entering At 1 sec
5
Enter The Size Of The Packet Entering At 1 sec
```

4
Enter The Size Of The Packet Entering At 1 sec
3
Second|Packet Recieved|Packet Sent|Packet Left|Packet Dropped|

1	5	2	3	0
2	4	2	3	2
3	3	2	3	1
4	0	2	1	0
5	0	1	0	0

VIVA QUESTIONS

1. What are functions of different layers?
2. Differentiate between TCP/IP Layers and OSI Layers
3. Why header is required?
4. What is the use of adding header and trailer to frames?
5. What is encapsulation?
6. Why fragmentation requires?
7. What is MTU?
8. Which layer imposes MTU?
9. Differentiate between flow control and congestion control.
10. Differentiate between Point-to-Point Connection and End-to-End connections.
11. What are protocols running in different layers?
12. What is Protocol Stack?
13. Differentiate between TCP and UDP.
14. Differentiate between Connectionless and connection oriented connection.
15. Why frame sorting is required?
16. What is meant by subnet?
17. What is meant by Gateway?
18. What is an IP address?
19. What is MAC address?
20. Why IP address is required when we have MAC address?
21. What is meant by port?
22. What are ephemerical port number and well known port numbers?
23. What is a socket?
24. What are the parameters of socket()?
25. Describe bind(), listen(), accept(),connect(), send() and recv().
26. What are system calls? Mention few of them.
27. What is IPC? Name three techniques.
28. Explain mkfifo(), open(), close() with parameters.
29. What is meant by file descriptor?
30. What is meant by traffic shaping?
31. How do you classify congestion control algorithms?
32. Differentiate between Leaky bucket and Token bucket.
33. How do you implement Leaky bucket?

34. How do you generate busty traffic?
35. What is the polynomial used in CRC-CCITT?
36. What are the other error detection algorithms?
37. What is difference between CRC and Hamming code?
38. Why Hamming code is called 7,4 code?
39. What is odd parity and even parity?
40. What is meant by syndrome?
41. What is generator matrix?
42. What is spanning tree?
43. Differentiate between Prim's and Kruskal's algorithm.
44. What are Routing algorithms?
45. How do you classify routing algorithms? Give examples for each.
46. What are drawbacks in distance vector algorithm?
47. How routers update distances to each of its neighbor?
48. How do you overcome count to infinity problem?
49. What is cryptography?
50. How do you classify cryptographic algorithms?
51. What is public key?
52. What is private key?
53. What are key, ciphertext and plaintext?
54. What is simulation?
55. What are advantages of simulation?
56. Differentiate between Simulation and Emulation.
57. What is meant by router?
58. What is meant by bridge?
59. What is meant by switch?
60. What is meant by hub?
61. Differentiate between route, bridge, switch and hub.
62. What is ping and telnet?
63. What is FTP?
64. What is BER?
65. What is meant by congestion window?
66. What is BSS?
67. What is incoming throughput and outgoing throughput?

68. What is collision?
69. How do you generate multiple traffics across different sender-receiver pairs?
70. How do you setup Ethernet LAN?
71. What is meant by mobile host?
72. What is meant by NCTUns?
73. What are dispatcher, coordinator and nctunsclient?
74. Name few other Network simulators
75. Differentiate between logical and physical address.
76. Which address gets affected if a system moves from one place to another place?
77. What is ICMP? What are uses of ICMP? Name few.
78. Which layer implements security for data?

REFERENCE

1. **Communication Networks: Fundamental Concepts and Key Architectures -** Alberto Leon, Garcia and Indra Widjaja, 3rd Edition, Tata McGraw- Hill, 2004.
2. **Data and Computer Communication,** William Stallings, 8th tEdition, Pearson Education, 2007.
3. **Computer Networks: A Systems Approach -** Larry L. Peterson and Bruce S. David, 4th Edition, Elsevier, 2007.
4. **Introduction to Data Communications and Networking** – Wayne Tomasi, Pearson Education, 2005.
5. **Communication Networks – Fundamental Concepts and Key architectures –** Alberto Leon- Garcia and Indra Widjaja:, 2rd Edition, Tata McGraw-Hill, 2004
6. **Computer and Communication Networks –** Nader F. Mir:, Pearson Education, 2007.