

Vidyavardhaka Sangha®, Mysore
VIDYAVARDHAKA COLLEGE OF ENGINEERING

Autonomous Institute, Affiliated to Visvesvaraya Technological University, Belagavi
(Approved by AICTE, New Delhi & Government of Karnataka)
Accredited by NBA | NAAC with 'A' Grade

Department of Information Science & Engineering

Phone: +91 821-4276210, Email: hodis@vvce.ac.in

Web: <http://www.vvce.ac.in>



   @vvceofficial

COMPUTER NETWORKS LABORATORY

21IS51

LAB MANUAL

Prepared by,

1. Dr. Vartika Sharma
2. Prof. Chayashree G

Commands to be used in LINUX environment:

- pwd:** To know which directory you are in
- ls:** to list all the files which are in the current directory
- cd:** change directory: cd src directory
- mkdir:** to create dir: mkdir dir name
- rmdir:** to delete a directory (to delete an empty directory). To delete a directory containing files, use **rm** to delete just the directory

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

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

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

```
import java.util.*;
public class CRC16CCITT
{
    public static void main(String[] args)
    {
        int crc = 0xFFFF; // initial value
        int polynomial = 0x1021; // 0001 0000 0010 0001
        byte[] bytes = args[0].getBytes();
        for (byte b : bytes)
        {
            for (int i = 0; i < 8; i++)
            {
                boolean bit = ((b >> (7-i) & 1) == 1);
                boolean c15 = ((crc >> 15 & 1) == 1);
                crc <<= 1;
                if (c15 ^ bit) crc ^= polynomial;
            }
        }
        crc &= 0xffff;
        System.out.println("Codeword received is = " + Integer.toHexString(crc));
        String fcrc= Integer.toHexString(crc);
        System.out.println("enter the received data ");
        Scanner sc=new Scanner(System.in);
        String a=sc.nextLine();
    }
}
```

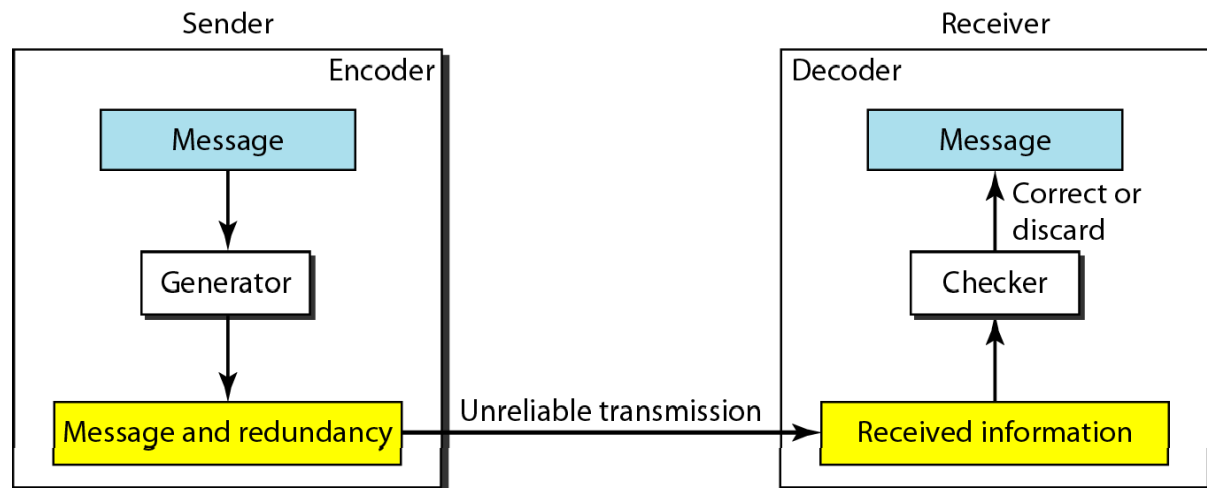
```
int deci=Integer.parseInt(a,16);
if((deci&=0xffff)==crc)
    System.out.println("received is correct");
else
    System.out.println("received is incorrect");
}
}
```

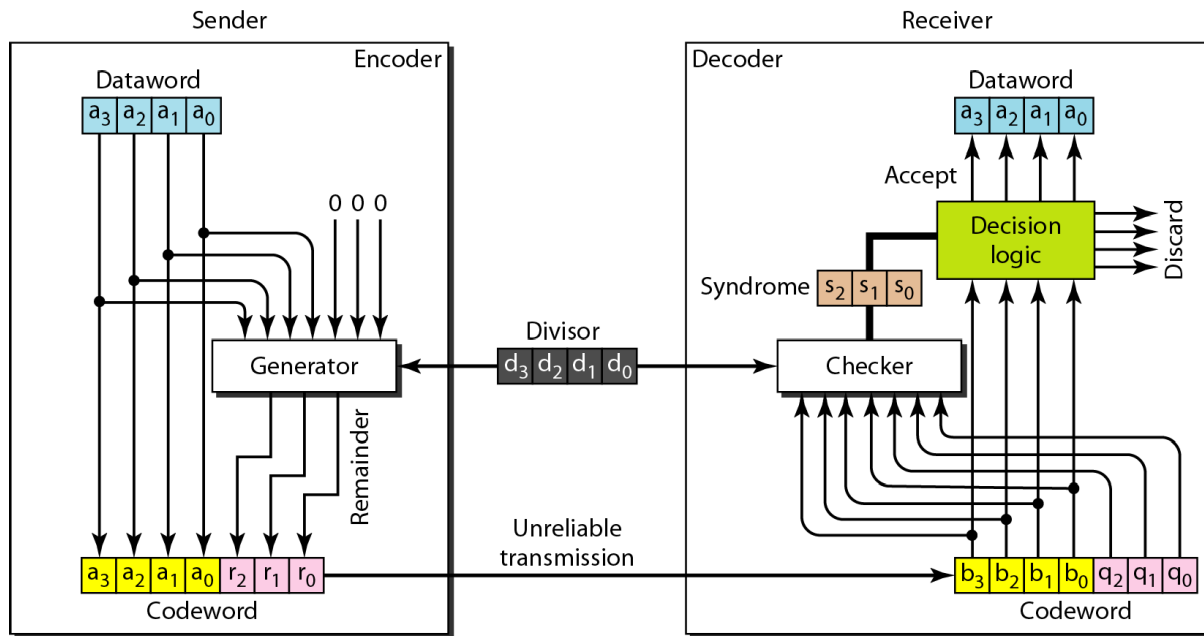
Output:

```
[student@localhost ~]$ vi CRC.java
[student@localhost ~]$ javac CRC.java
[student@localhost ~]$ java CRC b
code word received is=ad14
enter the recieved data
ad14
recieved is correct
[student@localhost ~]$ javac CRC.java
[student@localhost ~]$ java CRC b
code word received is=ad14
enter the recieved data
ac13
error detected
```

OR

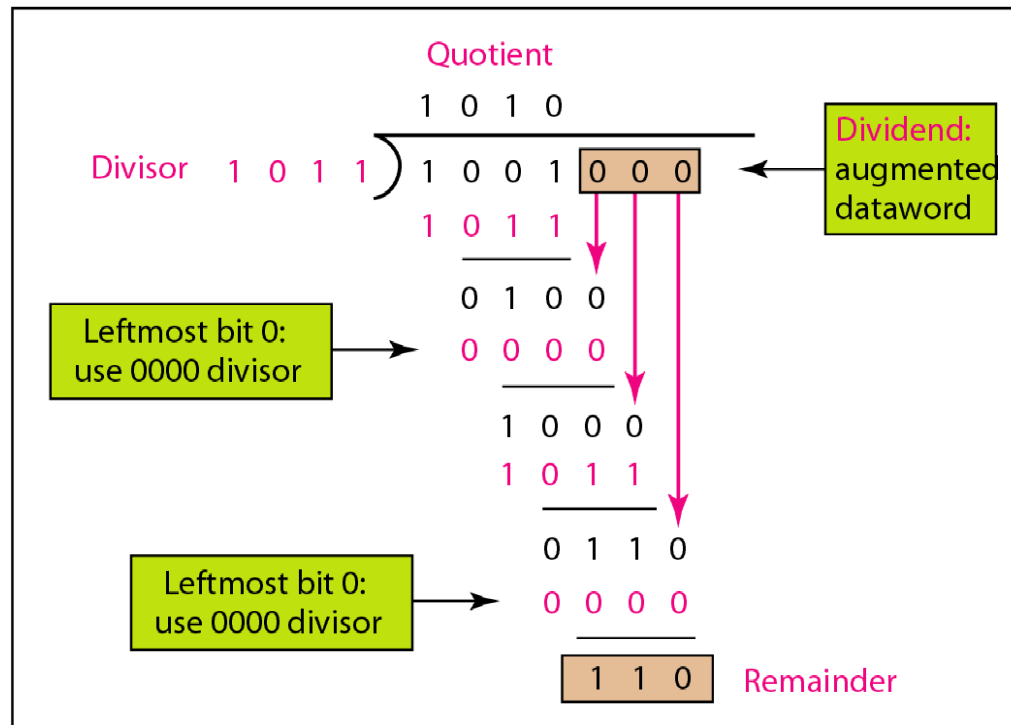
Write a program for error detecting code using CRC-CCITT (16- bits).



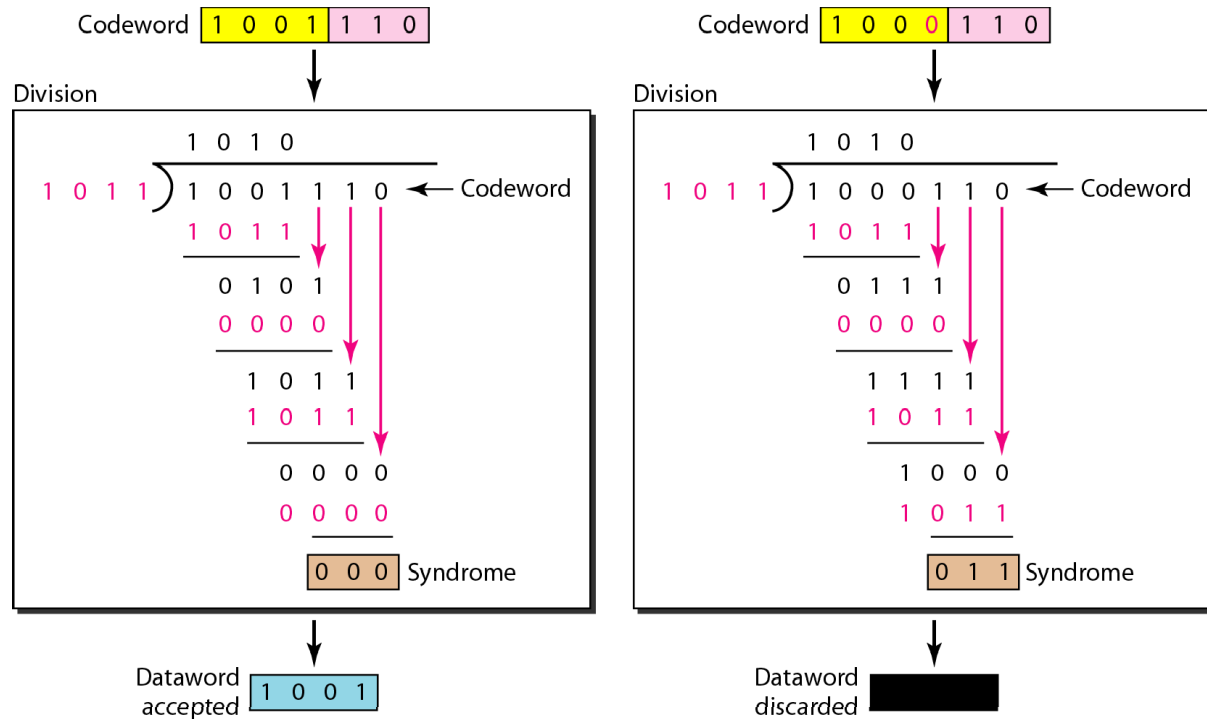


Dataword 1 0 0 1

Division



Codeword 1 0 0 1 1 1 0
Dataword Remainder



```

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; 20-16
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)
    {
        flag=1;
        break;
    }
}
if(flag==1)
    System.out.println("error in data");
else
    System.out.println("no error");
}
}

```

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|
| a | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|
| b | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|
| a | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |

CODEWORD

Dataword (n size): 1011

Divisor(generator polynomial –k-17bit): $x^{16}+x^{12}+x^5+1=10001000000100001$

r= k-1 number of zeros (checksum bits/redundant bits) augmented to data word to get dividend

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 |

Codeword=augment the final remainder from division to n data word.

| | | |
|---------------|----------------------------|-------------------------------|
| Dataword=1011 | Remainder=1011000101101011 | CODEWORD=10111011000101101011 |
|---------------|----------------------------|-------------------------------|

1011

10001000000100001) 10110000000000000000

10001000000100001

001110000001000010

000000000000000000

0011100000010000100

10001000000100001

00011010000101001010

10001000000100001

00001011000101101011

Receiver side error detection

Remainder

10001000000100001) 10111011000101101011

10001000000100001

001100110000011000

000000000000000000

0011001100000110001

10001000000100001

00010001000000100001

10001000000100001

000000000000000000

All zero's= no error

1011
10001000000100001) 10111010000100101011
10001000000100001
001100100000001000
0000000000000000
0011001000000010001
10001000000100001
0001000000001100001
10001000000100001
00000001000001000000

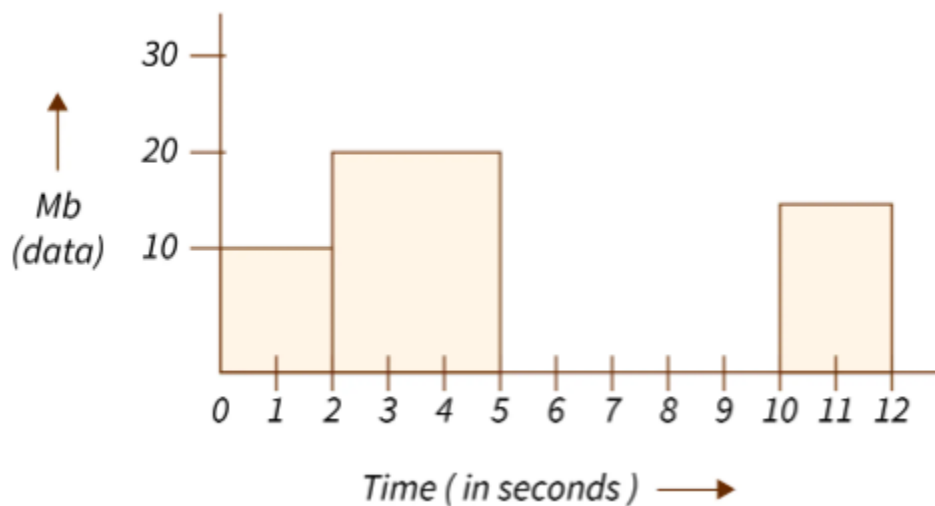
Few bits got modified=
error

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

Aim: control the rate at which traffic is sent to the network and shape the burst traffic to a steady traffic stream.

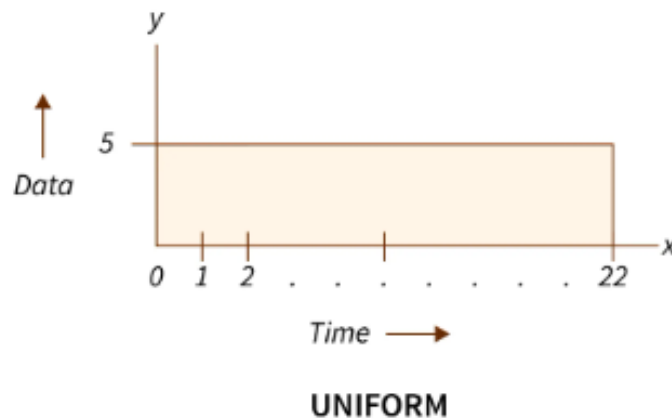
Let us say Host A sends:

- 10 Mbps data for the first 2 seconds.
- 20 Mbps data for the next 3 seconds.
- No data for the next 3 seconds.
- 15 Mbps data for the next 2 seconds.



The above traffic nature is called bursty traffic, coming from a source into the network. Bursty traffic is a sudden, unexpected network volume traffic peak, and depression. It is the data that is transmitted in a short, uneven manner. It can affect the network adversely and can lead to network congestion.

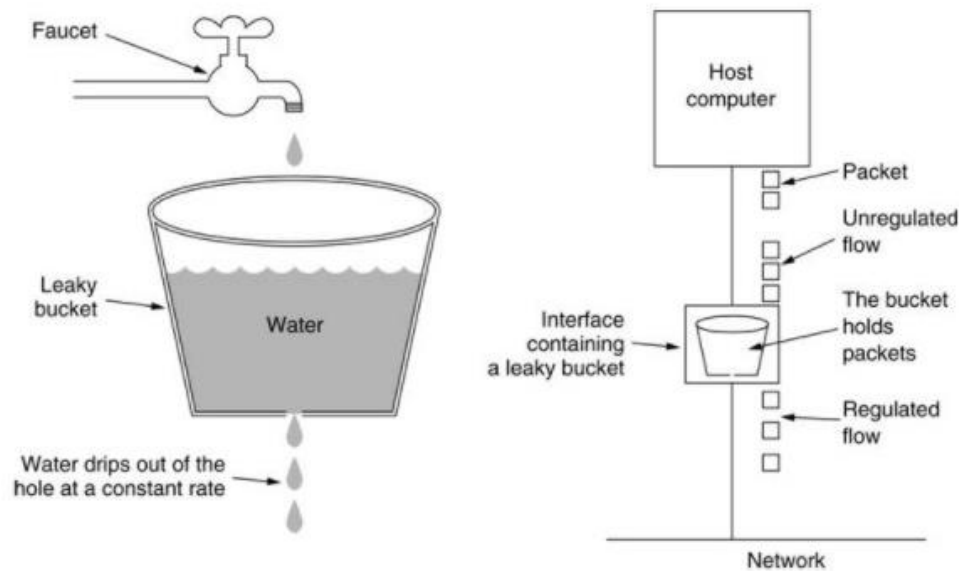
If many hosts are connected to a network and they keep sending bursty traffic to the network, it can lead to packet loss, and unexpected delays which decrease the quality of service, and packets can get backlogged. This unexpected packet delay and packet loss are termed **network congestion**.



Traffic Shaping is a mechanism to control the amount of traffic sent to the network. It means regulating the average rate flow of data to the network to prevent congestion. It maintains a constant traffic pattern that is sent to the network. This constant rate can depend on the network bandwidth limits and the priority of packets sent by the host.

The **Leaky Bucket algorithm** is a traffic shaping algorithm that is used to convert bursty traffic into smooth traffic by averaging the data rate sent into the network.

It is a method of congestion control where multiple packets are stored temporarily. These packets are sent to the network at a constant rate that is decided between the sender and the network.



- A state occurring in network layer when the message traffic is so heavy that it slows down network response time. The presence of congestion means that the load is greater than the resources can handle.
- Effects of Congestion
 - As delay increases, performance decreases.
 - If delay increases, retransmission occurs, making situation worse.
- There are two popular congestion control algorithm, and they are as follows:
 - Leaky Bucket Algorithm
 - Token Bucket Algorithm
- Leaky Bucket Algorithm is an congestion control method.
- The main concept of the leaky bucket algorithm is that the output data flow remains constant despite the variant input traffic
- 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).

Drawback

- 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 consider 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.

Steps involved in leaky bucket algorithm:

- When host wants to send packet, packet is thrown into the bucket.
- A packet with a size smaller than size of bucket arrives than that will be forward it.
- If the packet's size increases by more than size of bucket, it will either be discarded or queued.
- The bucket leaks at a constant rate, meaning the network interface transmits packets at a constant rate.
- Bursty traffic is converted to a uniform traffic by the leaky bucket.
- In practice the bucket is a finite queue that outputs at a finite rate.

```
=====
=====*****=====
```

Program:

```
import java.util.*;

class Leaky
{
private static int bucket_capacity, array_size, current_bucket, fixed_data_flow;
private static int array [];
int rem=0;

public void leakyworking (int a)
{
    if(a<=bucket_capacity)
    {
        //System.out.print("\t\t"+a);
        if(a<=fixed_data_flow)
        {
            System.out.print("\t\t"+a);
            current_bucket=0;
            System.out.println("\t\t"+current_bucket);
        }
        else
        {
            current_bucket=a-fixed_data_flow;
            System.out.print("\t\t"+(a-current_bucket));
        }
    }
}
```

```
        System.out.println("\t\t"+current_bucket);
    }
}
else
{
    System.out.println("\t\tDropped\t\t");
}
rem=current_bucket;
}

public void LeakyBucket()
{
    current_bucket=0;
    System.out.print("\npSize \tBucket \tPkt_snd \tRemaining\n" );
    for(int i=0;i<array_size;i++)
    {
        int input=array[i];
        System.out.print(input);
        if(rem!=0)
        {
            current_bucket=rem+input;
            System.out.print("\t"+current_bucket);
            leakyworking(current_bucket);
        }
        else
        {
            System.out.print("\t"+input);
            leakyworking(input);
        }
    }
}

public static void main(String args[])
{
    Leaky pr = new Leaky();
    Scanner scan = new Scanner(System.in);
    System.out.print("Enter the Bucket Capacity : ");
    bucket_capacity = scan.nextInt();
    System.out.print("\nEnter the Bucket Fixed Data Flow : ");
    fixed_data_flow = scan.nextInt();
    System.out.print("\nEnter the No. of packets : ");
    array_size=scan.nextInt();
    System.out.println("\nEnter the Input values of size : "+array_size);
    array = new int[array_size];
    for(int i=0;i<array_size;i++)
    {
        array[i]=scan.nextInt();
    }
    pr.LeadyBucket();
    System.out.print("\n\nPROGRAM TERMINATING SUCCESSFULLY...\n");
}
```

```
        scan.close();
    }
}
```

OUTPUT:

Enter the Bucket Capacity: 4

Enter the Bucket Fixed Data Flow: 3

Enter the No. of packets: 5

Enter the Input values of size: 5

2
4
1
5
3

| pSize | Bucket | Pkt_snd | Remaining |
|-------|--------|---------|-----------|
| 2 | 2 | 2 | 0 |
| 4 | 4 | 3 | 1 |
| 1 | 2 | 2 | 0 |
| 5 | 5 | Dropped | |
| 3 | 3 | 3 | 0 |

PROGRAM TERMINATING SUCCESSFULLY...

[OR]

```
import java.util.Scanner;
public class Leaky
{
    public static int bucketSize = 1000;
    public static int outputRate = 100;
    public static void sendPacket(int pktSize)
    {
        if (pktSize > bucketSize)
        {
            System.out.println("Bucket OverFlow");
        }
        else
        {

```

```
        while (pktSize > outputRate)
        {
            System.out.println(outputRate + " bytes of packet is sent");
            pktSize = pktSize - outputRate;
        }
        System.out.println(pktSize + " bytes of packet is sent");
    }
}

public static void main(String[] args)
{
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter the no of packets: ");
    int numpackets = scanner.nextInt();
    if (numpackets > 0)
    {
        for (int i = 1; i <= numpackets; i++)
        {
            System.out.print("Enter the packet " + i + " size : ");
            int pktSize = scanner.nextInt();
            sendPacket(pktSize);
        }
    }
    else
    {
        System.out.println("No Packets to Send");
    }
}
}
```

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

RSA algorithm is an asymmetric cryptography algorithm. Asymmetric means that it works on two different keys i.e. **Public Key** and **Private Key**.

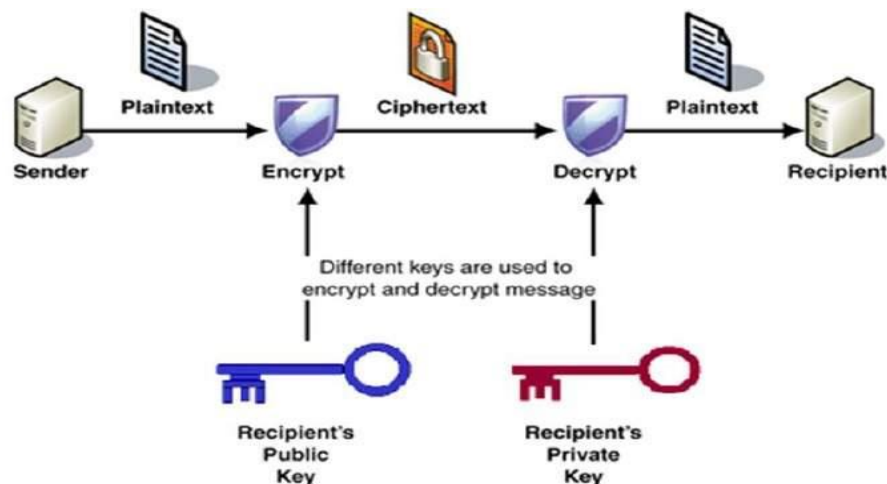
Network security is concerned mainly with the following two elements:

1. **Confidentiality**- Information should be available only to those who have rightful access to it.
2. **Authenticity and integrity**- The sender of a message and the message itself should be verified at the receiving point.

Computer communication networks can use following common solutions to protect from attacks are:

- Cryptographic Techniques
- Authentication techniques

Cryptographic Techniques is the process of transforming a piece of information or message shared by two parties into some sort of code.



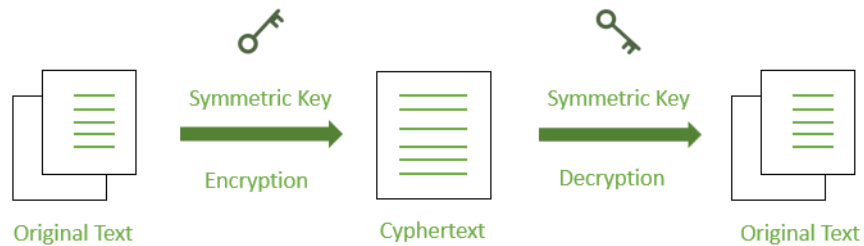
The two types of encryption techniques are

- Secret key encryption (**Symmetric key Cryptography**)
- Public key encryption

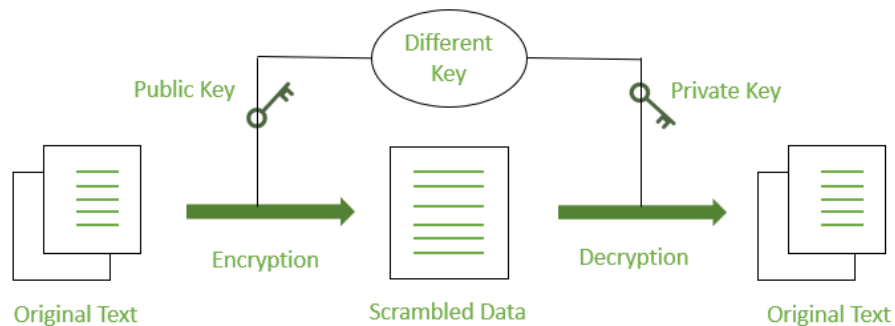
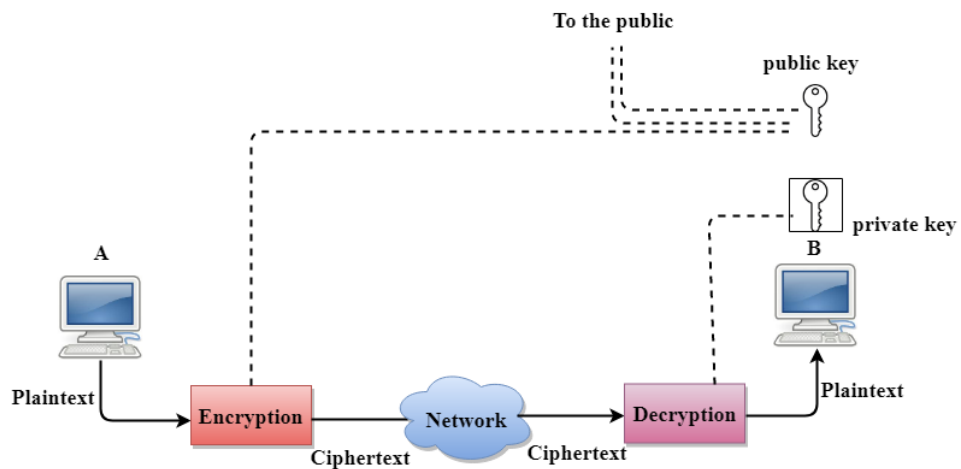
Symmetric key Cryptography

- It is also called *Secret key encryption* protocols or *single key encryption* protocols.
- The same key is used by both the parties. The sender uses the secret key and encryption algorithm to encrypt the data; the receiver uses this key and decryption algorithm to decrypt the data.
- **Data Encryption Standard (DES)** and **Advanced Encryption Standard (AES)**.

- It consists of an encryption algorithm, a key, and a decryption algorithm.
- The encrypted message is called *ciphertext*.
- The encryption algorithm need not be kept secret; only the key must be secret.



Public Key Encryption/Decryption technique

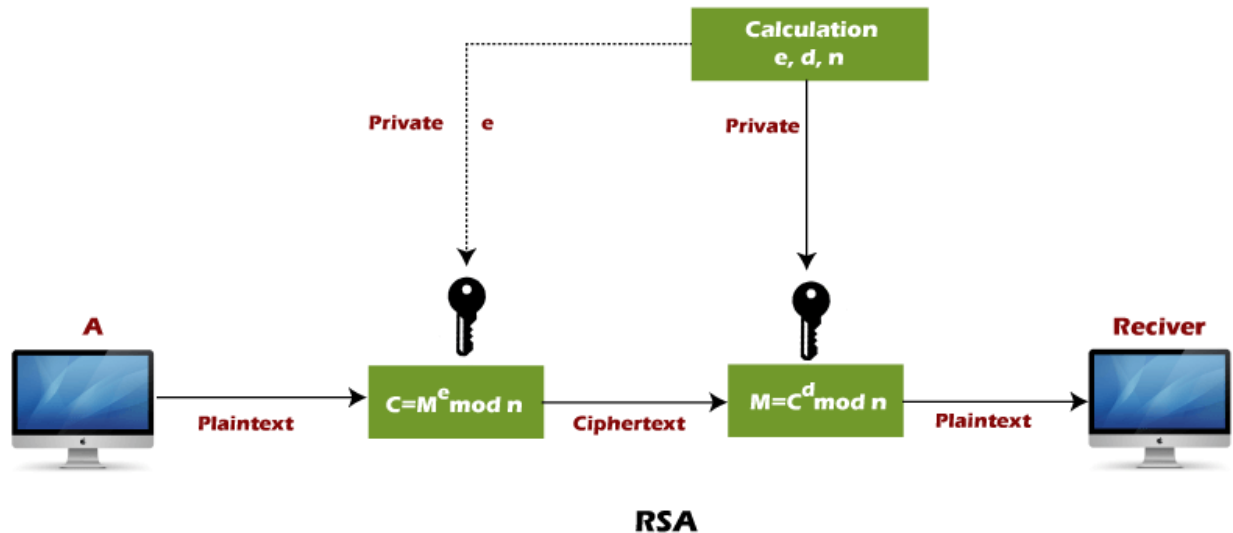


- There are two keys in public key encryption: a private key and a public key.

- The private key is given to the receiver while the public key is provided to the public.

RSA (Rivert, Shamir, and Aldeman) Protocol

RSA algorithm is an asymmetric cryptography algorithm.



key generation

- **Generate the RSA modulus (n)**
 - Select two large primes, p and q.
 - Calculate $n = p * q$. For strong unbreakable encryption, let n be a large number, typically a minimum of 512 bits.
- **Find Derived Number (e) public key**
 - Number e must be greater than 1 and less than $(p - 1)(q - 1)$.
 - There must be no common factor for e and $(p - 1)(q - 1)$ except for 1. In other words two numbers e and $(p - 1)(q - 1)$ are coprime.
- **Form the public key**
 - The pair of numbers (n, e) form the RSA public key and is made public.
- **Generate the private key**
 - Private Key d is calculated from p, q, and e.
$$d = e \text{ modulo } (p - 1)(q - 1) \text{ [OR] } ed = 1 \bmod (p - 1)(q - 1)$$

It can be written as: $d = (((p-1)(q-1))*i + 1)/e$

- Public key = {e, n} and private key = {d, n}

Encryption

- Suppose the sender wish to send some text message to someone whose public key is (n, e).
- The sender then represents the plaintext as a series of numbers less than n.
- To encrypt the first plaintext P, which is a number modulo n. The encryption process is simple mathematical step as $C = P^e \bmod n$ where, $P < n$

C=Cipher text, P=plain text, e= encryption key and n=block size

Decryption

- Suppose that the receiver of public-key pair (n, e) has received a ciphertext C.

Receiver raises C to the power of his private key d. The result modulo n will be the plaintext

$$P = C^d \bmod n$$

Algorithm

- Step 1.** Choose two prime numbers p and q.
- Step 2.** Calculate $n = p * q$
- Step 3.** Calculate $\phi(n) = (p - 1) * (q - 1)$
- Step 4.** Choose e such that $\gcd(e, \phi(n)) = 1$
- Step 5.** Calculate d such that $e * d \bmod \phi(n) = 1$ [or] $d = e \text{ modulo } (p - 1)(q - 1)$
 $d = (((\phi(n)) * i) + 1) / e$
- Step 6.** Public Key {e, n} Private Key {d, n}
- Step 7.** Cipher text $C = P^e \bmod n$ where P = plaintext
- Step 8.** For Decryption $D = C^d \bmod n$ where D will give back the plaintext

1. Select primes $p = 11, q = 3$.
2. $n = pq = 11 * 3 = 33$
3. $\phi(n) = (p-1)(q-1) = 10.2 = 20$
4. Choose $e=3$
Check $\gcd(e, \phi(n)) = \gcd(3, 20) = 1$
5. Compute d such that $ed \equiv 1 \pmod{\phi(n)}$
i.e. compute $d = e^{-1} \bmod \phi(n)$.
 $d = (((\phi(n)) * i) + 1) / e$
6. Public key = (n, e) = (33, 3) Private Key = (n, d) = (33, 7).

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

$$\begin{aligned} c &= m^e \bmod n \\ &= 7^3 \bmod 33 \end{aligned}$$

$$= 343 \bmod 33$$

= 13. Hence the ciphertext $c = 13$.

To check **decryption** we compute

$$\begin{aligned} m' &= c^d \bmod n \\ &= 13^7 \bmod 33 \\ &= 7. \end{aligned}$$

```
import java.util.Scanner;
public class Rsa
{
    public static int mult(int x,int y,int n)
    {
        int k=1, j;
        for(j=1;j<=y;j++)
            k=(k*x)%n;
        return k;
    }
    public static int gcd(int m,int n)
    {
        if(n==0)
            return m;
        else
            return (gcd(n,m%n));
    }
    public static void main(String[] args)
    {
        int msg,plaintext,ciphertext;
        int n,d=0,e,z,p,q,i;
        Scanner scanner = new Scanner(System.in);
        System.out.println("enter two values p and q: ");
        p = scanner.nextInt();
        q = scanner.nextInt();

        System.out.println("enter message ");
        msg=scanner.nextInt();
        n=p*q;
        z=(p-1)*(q-1);
        do
```

```
{
    System.out.print("choose the value of e (e<2) such that gcd(z,e)=1: ");
    e=scanner.nextInt();
} while (gcd(z,e)!=1);
i=2;
while (((i*e)%z)!=1)
{
    i++;
    d=i;
}
System.out.println("The public key pair is (" + e + "," + n + ")");
System.out.println("The private key pair is (" + d + "," + n + ")");
ciphertext = mult(msg,e,n);
System.out.println("cipher text = " + ciphertext);
plaintext = mult(ciphertext,d,n);
System.out.println("plain text = " + plaintext);
}
```

```
[student@localhost ~]$ vi Rsa.java
[student@localhost ~]$ javac Rsa.java
[student@localhost ~]$ java Rsa
enter two values p and q:
2
4
2is a prime number
4not a prime number
[student@localhost ~]$ java Rsa
enter two values p and q:
3
13
3is a prime number
13is a prime number
enter message
10
choose the value of e (e<2) such that gcd(z,e)=1: 5
The public key pair is (5,39)
The private key pair is (5,39)
cipher text = 4
plain text = 10
[student@localhost ~]$
```

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.

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.

The following steps occur when establishing a TCP connection between two computers using sockets:

- The server instantiates a ServerSocket object, denoting which port number communication is to occur on.
- The server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port.
- After the server is waiting, a client instantiates a Socket object, specifying the server's name and the port number to connect to.
- The constructor of the Socket class attempts to connect the client to the specified server and the port number. If communication is established, the client now has a Socket object capable of communicating with the server.
- On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket.

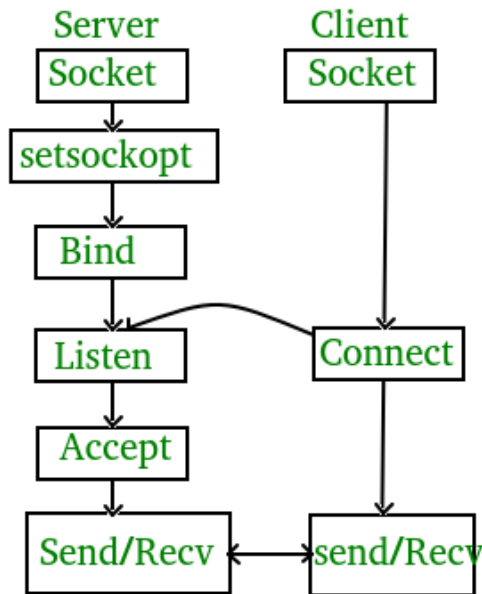
After the connections are established, communication can occur using I/O streams. Each socket has both an OutputStream and an InputStream. The client's OutputStream is connected to the server's InputStream, and the client's InputStream is connected to the server's OutputStream.

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 the required file using send () system call.
5. Read the contents of the file sent by the server by recv() system call.
6. Stop.

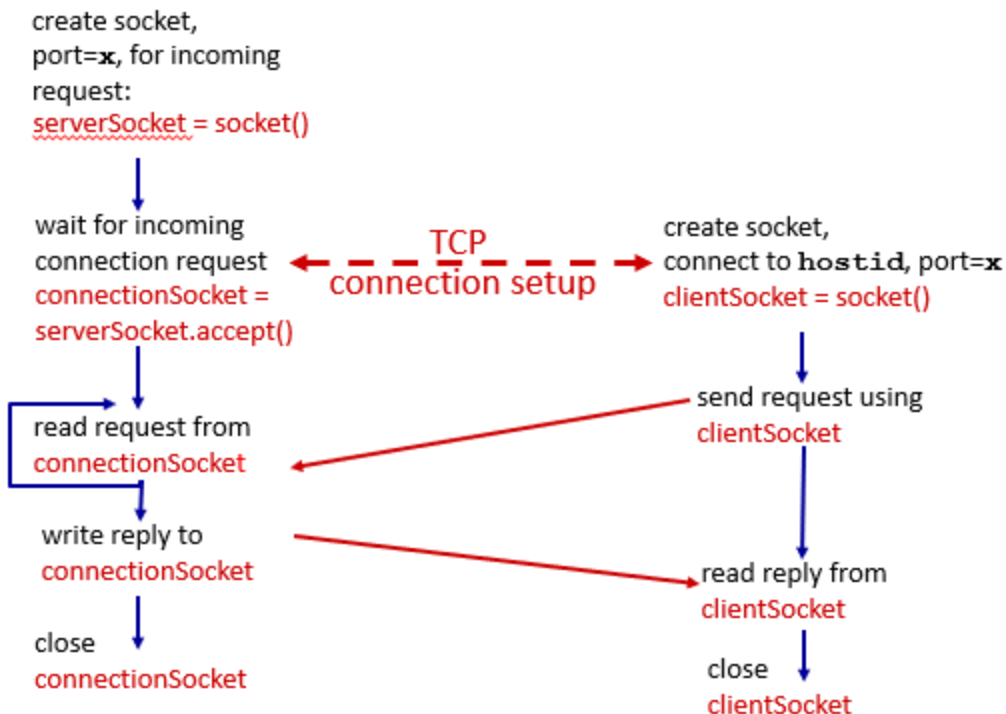
Algorithm: Server Side

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.



server (running on hostid)

client



A stream is a sequence of data. In Java, a stream is composed of bytes.

output stream to write data to a destination; it may be a file, an array, peripheral device or socket.

input stream to read data from a source; it may be a file, an array, peripheral device or socket.

TCP Client

```
import java.io.*;
```

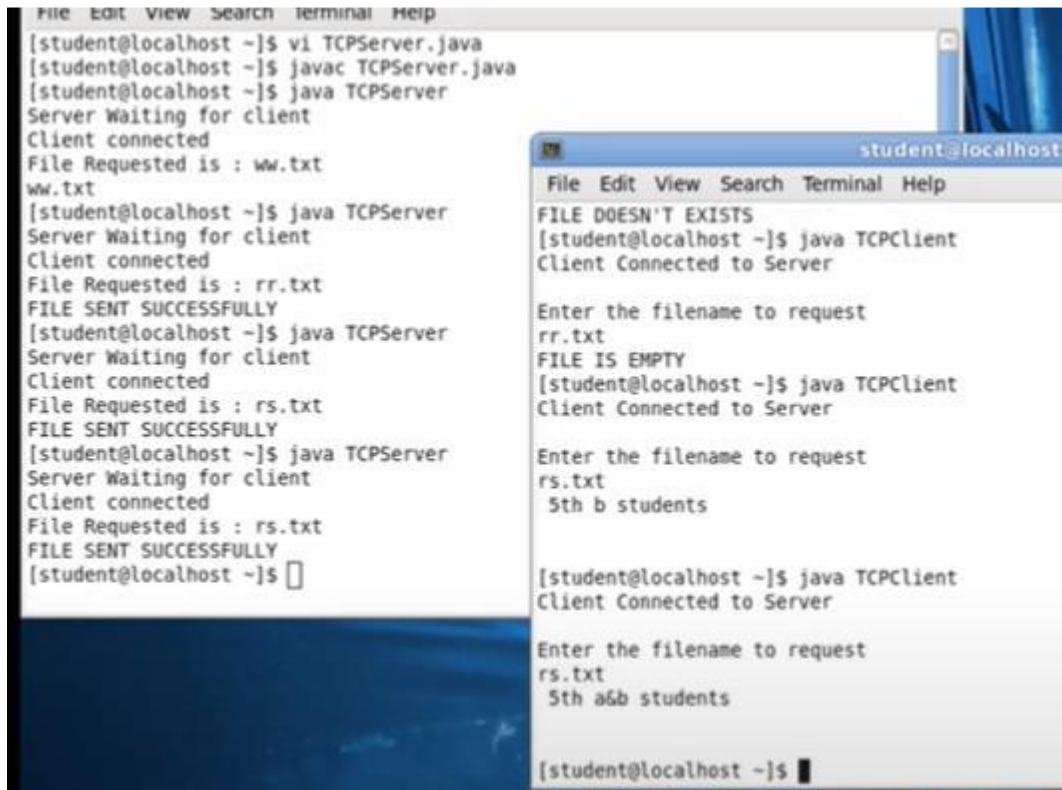
```
import java.net.*;
import java.util.Scanner;
public class TCPClient
{
    public static void main(String[] args) throws IOException, InterruptedException
    {
        DataOutputStream out;
        DataInputStream in;
        Scanner scanner = new Scanner(System.in);
        Socket socket = new Socket("127.0.0.1", 6000); //server IP and Port num
        System.out.println("Client Connected to Server");
        System.out.print("\nEnter the filename to request\n");
        String filename = scanner.nextLine();
        in = new DataInputStream(socket.getInputStream()); // it will read the data from the source
        out = new DataOutputStream(socket.getOutputStream()); // it will write the data into the
        destination
        out.writeUTF(filename); // Unicode transformation format
        String fileContent = in.readUTF();
        if (fileContent.length() > 0)
            System.out.println(fileContent);
        else
            System.out.println("FILE IS EMPTY");
    }
}
```

At server side:

```
import java.io.*;
import java.net.*;
import java.nio.file.*;

public class TCPServer
{
    public static void main(String[] args) throws IOException
    {
        ServerSocket server;
        DataOutputStream out = null;
        DataInputStream in;
        try
        {
            server = new ServerSocket(6000, 1); //port number and num of connections
            System.out.println("Server Waiting for client");
            Socket socket = server.accept();
            System.out.println("Client connected ");
        }
    }
}
```

```
        in = new DataInputStream(socket.getInputStream());
        out = new DataOutputStream(socket.getOutputStream());
        String fileName = in.readUTF();
        System.out.println("File Requested is : " + fileName);
        byte[] filedata = Files.readAllBytes(Paths.get(fileName));
        String fileContent = new String(filedata);
        out.writeUTF(fileContent.toString());
        System.out.println("FILE SENT SUCCESSFULLY");
    }
    catch (Exception e)
    {
        System.out.println(e.getMessage());
        out.writeUTF("FILE DOESN'T EXISTS");
    }
}
```



```
File Edit View Search Terminal Help
[student@localhost ~]$ vi TCPServer.java
[student@localhost ~]$ javac TCPServer.java
[student@localhost ~]$ java TCPServer
Server Waiting for client
Client connected
File Requested is : ww.txt
ww.txt
[student@localhost ~]$ java TCPServer
Server Waiting for client
Client connected
File Requested is : rr.txt
FILE SENT SUCCESSFULLY
[student@localhost ~]$ java TCPServer
Server Waiting for client
Client connected
File Requested is : rs.txt
FILE SENT SUCCESSFULLY
[student@localhost ~]$ java TCPServer
Server Waiting for client
Client connected
File Requested is : rs.txt
FILE SENT SUCCESSFULLY
[student@localhost ~]$

student@localhost:~$
File Edit View Search Terminal Help
FILE DOESN'T EXISTS
[student@localhost ~]$ java TCPClient
Client Connected to Server

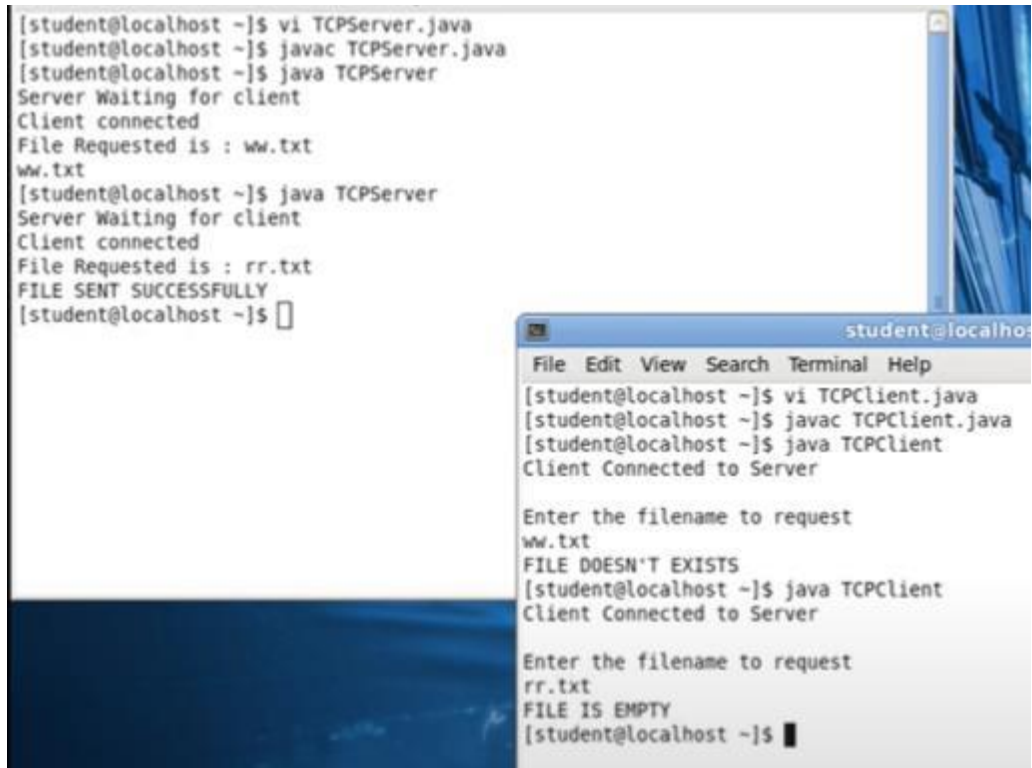
Enter the filename to request
rr.txt
FILE IS EMPTY
[student@localhost ~]$ java TCPClient
Client Connected to Server

Enter the filename to request
rs.txt
5th b students

[student@localhost ~]$ java TCPClient
Client Connected to Server

Enter the filename to request
rs.txt
5th a&b students

[student@localhost ~]$
```

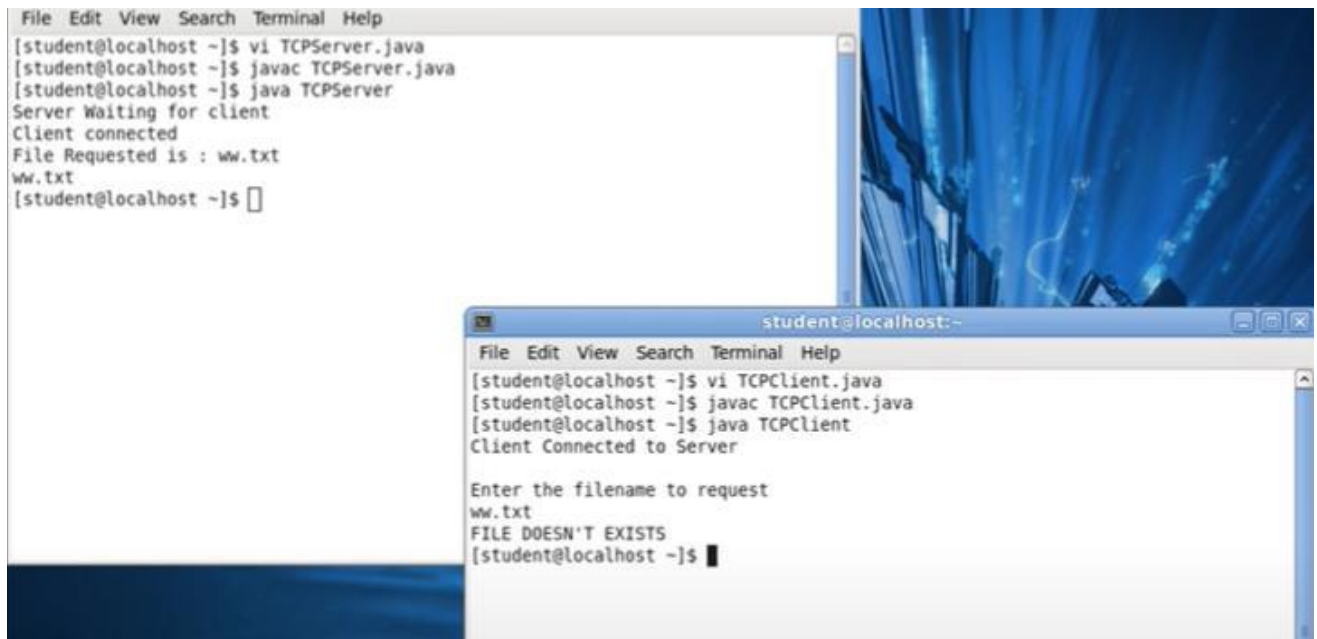
The image shows two terminal windows. The left window is the server's terminal, and the right window is the client's terminal. Both are running on a system with a blue desktop background.

```
[student@localhost ~]$ vi TCPServer.java
[student@localhost ~]$ javac TCPServer.java
[student@localhost ~]$ java TCPServer
Server Waiting for client
Client connected
File Requested is : ww.txt
ww.txt
[student@localhost ~]$ java TCPServer
Server Waiting for client
Client connected
File Requested is : rr.txt
FILE SENT SUCCESSFULLY
[student@localhost ~]$
```

```
student@localhost:~
File Edit View Search Terminal Help
[student@localhost ~]$ vi TCPClient.java
[student@localhost ~]$ javac TCPClient.java
[student@localhost ~]$ java TCPClient
Client Connected to Server

Enter the filename to request
ww.txt
FILE DOESN'T EXISTS
[student@localhost ~]$ java TCPClient
Client Connected to Server

Enter the filename to request
rr.txt
FILE IS EMPTY
[student@localhost ~]$
```



This image is another instance of the previous screenshot, showing the same terminal interactions for the TCP server and client. The server terminal shows the same sequence of commands and outputs, and the client terminal also shows the same sequence of commands and outputs.

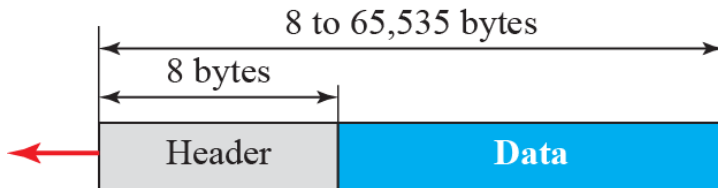
```
File Edit View Search Terminal Help
[student@localhost ~]$ vi TCPServer.java
[student@localhost ~]$ javac TCPServer.java
[student@localhost ~]$ java TCPServer
Server Waiting for client
Client connected
File Requested is : ww.txt
ww.txt
[student@localhost ~]$
```

```
student@localhost:~
File Edit View Search Terminal Help
[student@localhost ~]$ vi TCPClient.java
[student@localhost ~]$ javac TCPClient.java
[student@localhost ~]$ java TCPClient
Client Connected to Server

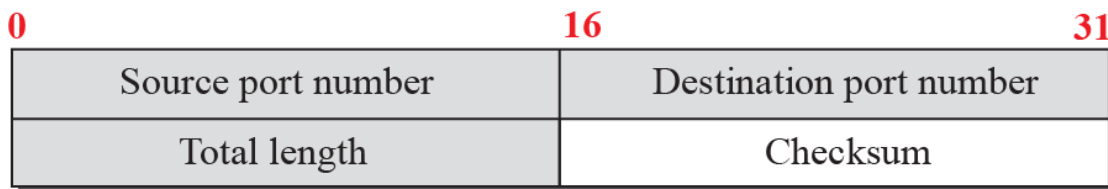
Enter the filename to request
ww.txt
FILE DOESN'T EXISTS
[student@localhost ~]$
```

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

UDP provides process to process communication using socket addresses

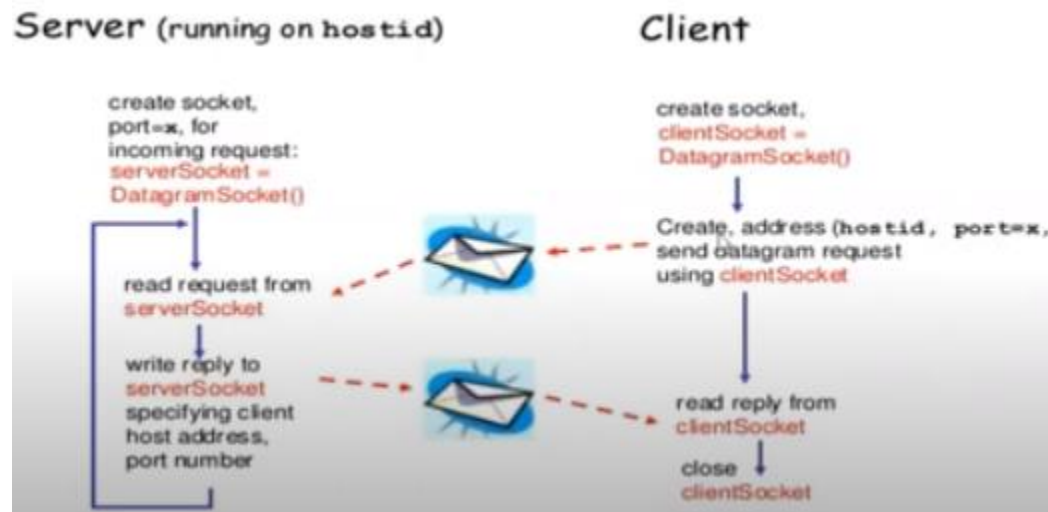


a. UDP user datagram



b. Header format

- UDP provides unreliable data transfer
- No connection between client and server
- No handshaking
- Sender explicitly attaches IP address and port of destination to each packet
- Transmitted data may be received out of order or lost.



//UDP Client

```
import java.net.*;
public class UDPclient
```

```
{
    public static void main(String[] args) throws Exception
    {
        byte[] buf = new byte[1024];
        System.out.println("Receiver");
        DatagramSocket ds = new DatagramSocket(3000);
        while(true)
        {
            DatagramPacket dp = new DatagramPacket(buf, 1024);
            ds.receive(dp);
            String msg = new String(dp.getData(), 0, dp.getLength());
            System.out.println(msg);
        }
    }
}
```

// UDP Server

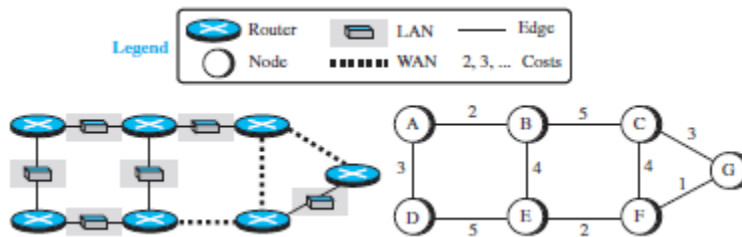
```
import java.net.*;
import java.util.Scanner;
public class UDPserver
{
    public static void main(String[] args) throws Exception
    {
        System.out.println("Sender");
        DatagramSocket ds = new DatagramSocket();
        Scanner scanner = new Scanner(System.in);
        System.out.println("\n Enter the Message : ");
        while(true)
        {
            String msg = scanner.nextLine();
            InetAddress ip = InetAddress.getByName("127.0.0.1");
            DatagramPacket dp = new DatagramPacket(msg.getBytes(),
            msg.length(), ip, 3000);
            ds.send(dp);
        }
    }
}
```

```
File Edit View Search Terminal Help
[root@localhost Desktop]# javac DReciever.java
[root@localhost Desktop]# java DReciever
Receiver
hello Good Morning
█
```

```
File Edit View Search Terminal Help
[root@localhost Desktop]# javac DSender.java
[root@localhost Desktop]# java DSender
Sender

Enter the Message :
hello Good Morning
█
```

Write a program to find the shortest path between vertices using Bellman-Ford algorithm.



Least Cost Routing: The best route from the source router to the destination router is to find the least cost between the two.

- **Forwarding(router-local action):** transfer of a packet from an incoming link to an outgoing link within a single router.
- **Routing(network-wide process):** The network layer must determine the route or path taken by packets as they flow from a sender to a receiver.

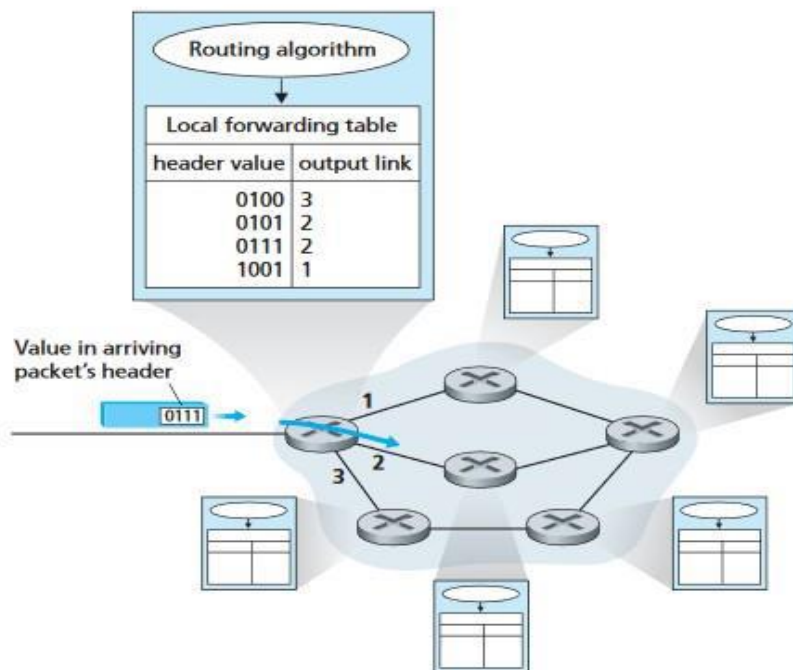


Figure 4.2 ♦ Routing algorithms determine values in forwarding tables

Least Cost Routing: The best route from the source router to the destination router is to find the least cost between the two.

A **least-cost tree** is a tree with the source router as the root that spans the whole graph (visits all other nodes) and in which the path between the root and any other node is the shortest. [or] It is a combination of least-cost paths from the root of the tree to all destinations.

Distance Vector Routing:

- Each node creates its own least-cost tree with the rudimentary information it has about its immediate neighbours.
- The incomplete trees are exchanged between immediate neighbours to make the trees more and more complete and to represent the whole internet.
- It uses Bellman-Ford equation to find the least cost.

$$D_{xy} = \min \{ (c_{xa} + D_{ay}), (c_{xb} + D_{by}), (c_{xc} + D_{cy}), \dots \}$$

- update an existing least cost with a newly identified least cost through an intermediary node

$$D_{xy} = \min \{ D_{xy}, (c_{xi} + D_{iy}) \}$$

A distance vector is a one-dimensional array to represent the tree.



Bellman–Ford algorithm

- The Bellman–Ford algorithm is an algorithm that computes shortest paths from a single source vertex to all 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
- 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

Algorithm

Following are the detailed steps.

Input: Graph and a source vertex *src*

Output: Shortest distance to all vertices from *src*. If there is a negative weight cycle, then shortest distances are not calculated, negative weight cycle is reported.

- This step initializes distances from the source to all vertices as infinite and distance to the source itself as 0. Create an array `dist[]` of size `|V|` with all values as infinite except `dist[src]` where `src` is source vertex.
- This step calculates shortest distances. Do following `|V|-1` times where `|V|` is the number of vertices in given graph.

-a) Do following for each edge u-v
.....If $\text{dist}[v] > \text{dist}[u] + \text{weight of edge uv}$, then update $\text{dist}[v]$
..... $\text{dist}[v] = \text{dist}[u] + \text{weight of edge uv}$
3. This step reports if there is a negative weight cycle in graph. Do following for each edge u-v
.....If $\text{dist}[v] > \text{dist}[u] + \text{weight of edge uv}$, then “Graph contains negative weight cycle”
The idea of step 3 is, step 2 guarantees the shortest distances if the graph doesn't contain a negative weight cycle. If we iterate through all edges one more time and get a shorter path for any vertex, then there is a negative weight cycle

Like other Dynamic Programming Problems, the algorithm calculates shortest paths in a bottom-up manner.

- It first calculates the shortest distances which have at-most one edge in the path.
- Then, it calculates the shortest paths with at-most 2 edges, and so on.
- After the i-th iteration of the outer loop, the shortest paths with at most i edges are calculated. There can be maximum $|V| - 1$ edges in any simple path, that is why the outer loop runs $|V| - 1$ times. The idea is, assuming that there is no negative weight cycle, if we have calculated shortest paths with at most i edges, then an iteration over all edges guarantees to give shortest path with at-most (i+1) edges

Write a program to find the shortest path between vertices using Bellman-Ford algorithm.

Program 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();
}
}
```

Output:

Enter the number of vertices

4

Enter the adjacency matrix

0 5 0 0

5 0 3 4

0 3 0 2

0 4 2 0

Enter the source vertex

2

Distance of source 2 to 1 is 5

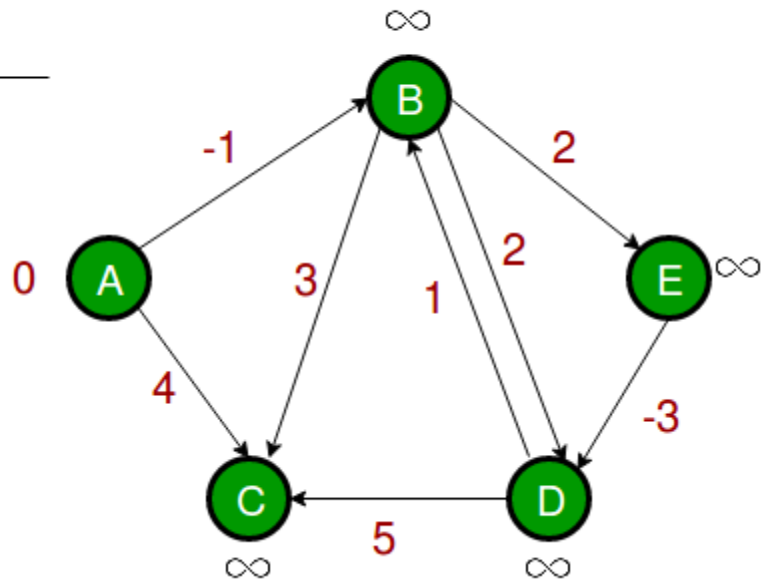
Distance of source 2 to 2 is 0

Distance of source 2 to 3 is 3

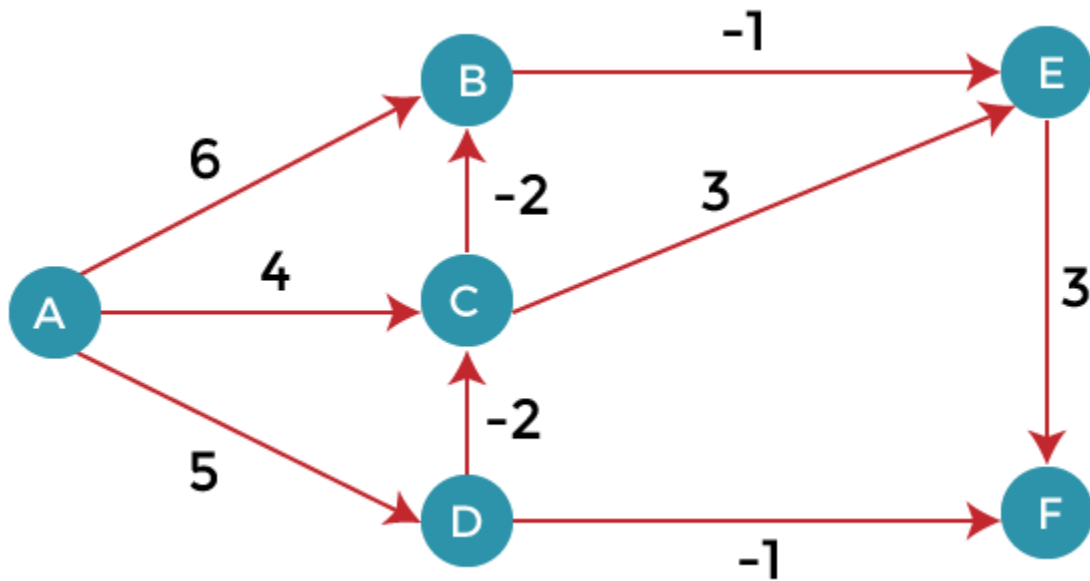
Distance of source 2 to 4 is 4

Input:2

| A | B | C | D | E |
|---|----------|----------|----------|----------|
| 0 | ∞ | ∞ | ∞ | ∞ |

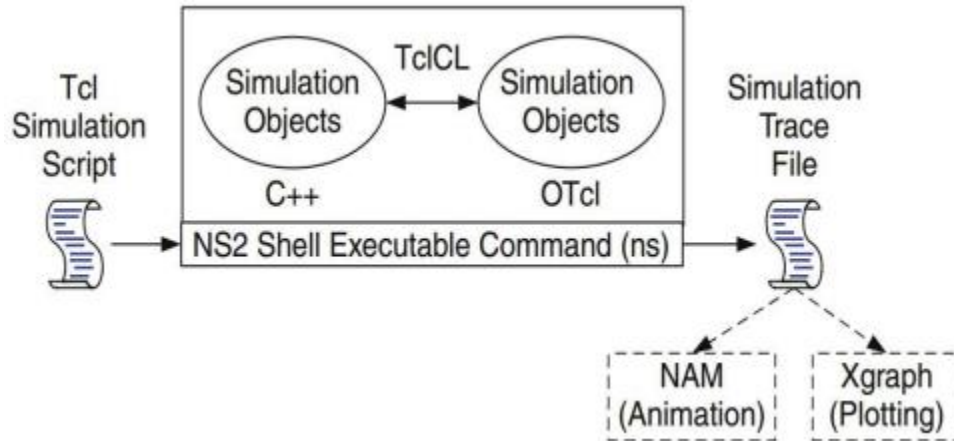


Input 3:



INTRODUCTION TO NS2

- 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 behaviours.



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
- **Loops**
while{\$i < \$n} {
..... }
for {set i 0} {\$i < \$n} {incr i} {
..... }
}

Wired TCL Script Components

- Create the event scheduler
- Open new files & turn on the tracing
- Create the nodes
- Setup the links
- Configure the traffic type (e.g., TCP, UDP, etc)
- Set the time of traffic generation (e.g., CBR, FTP)
- Terminate the simulation

NS Simulator Preliminaries.

1. Initialization and termination aspects of the ns simulator.
2. Definition of network nodes, links, queues and topology.
3. Definition of agents and of applications.
4. The nam visualization tool.
5. Tracing and random variables.

Initialization and Termination of TCL Script in NS-2

An ns simulation starts with the command: **set ns [new Simulator]** Which is thus the first line in the tcl script? This line declares a new variable as using the set command, you can call this variable as you wish, In general people declares it as ns because it is an instance of the Simulator class, so an object the code[new Simulator] is indeed the installation of the class Simulator using the reserved word new.

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

#Open the Trace file

```
set tracefile1 [open out.tr w]
```

```
$ns trace-all $tracefile1
```

#Open the NAM trace file

```
set namfile [open out.nam w]
```

```
$ns namtrace-all $namfile
```

The above creates a data 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 parameter the name of the file where the traces will go.

The last line tells the simulator to record all simulation traces in NAM input format. It also gives the file name that the trace will be written to later by the command `$ns flush-trace`. In our case, this will be the file pointed at by the pointer `—$namfile`, i.e the file `—out.tr`.

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).

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

$cbr attach-agent $udp

$cbr set packetsize_ 100

$cbr set rate_ 0.01Mb

$cbr set random_ false
```

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

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

Agents and Applications: We need to define routing (sources, destinations) the agents (protocols) the application that use them.

FTP over TCP: TCP is a dynamic reliable congestion control protocol. It uses Acknowledgements created by the destination to know whether packets are well received. There are number variants of the TCP protocol, such as Tahoe, Reno, NewReno, Vegas. The type of agent appears in the first line:

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

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

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

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

#Setup a UDP connection

```
set udp [new Agent/UDP]

$ns attach-agent $n1 $udp

set null [new Agent/Null]

$ns attach-agent $n5 $null

$ns connect $udp $null

$udp set fid_2
```

#setup a CBR over UDP connection

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

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.

```
set ns [new Simulator] /* Letter S is capital */ int a=0

set nf [open lab1.nam w] /* open a nam trace file in write mode */
$ns namtrace-all $nf /* nf – nam file */

set tf [open lab1.tr w] /* tf- trace file */
$ns trace-all $tf

proc finish { } { /* provide space b/w proc and finish and all are in small case */
global ns nf tf
$ns flush-trace /* clears trace file contents */
close $nf
close $tf
exec nam lab1.nam &
exit 0
}

set n0 [$ns node] /* creates 4 nodes */
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

$ns duplex-link $n0 $n2 200Mb 10ms DropTail /*Letter M is capital Mb*/
$ns duplex-link $n1 $n2 100Mb 5ms DropTail /*D and T are capital*/
$ns duplex-link $n2 $n3 1Mb 1000ms DropTail

$ns queue-limit $n0 $n2 10
$ns queue-limit $n1 $n2 10

set udp0 [new Agent/UDP] /* Letters A,U,D and P are capital */
$ns attach-agent $n0 $udp0

set cbr0 [new Application/Traffic/CBR] /* A,T,C,B and R are capital*/
$cbr0 set packetSize 500 /*S is capital, space after underscore*/
$cbr0 set interval 0.005
$cbr0 attach-agent $udp0

set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
```

```
set udp2 [new Agent/UDP]
$ns attach-agent $n2 $udp2
set cbr2 [new Application/Traffic/CBR]
$cbr2 attach-agent $udp2

set null0 [new Agent/Null] /* A and N are capital */
$ns attach-agent $n3 $null0

$ns connect $udp0 $null0
$ns connect $udp1 $null0

$ns at 0.1 "$cbr0 start"
$ns at 0.2 "$cbr1 start"
$ns at 1.0 "finish"
$ns run
```

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

/*immediately after BEGIN should open braces „{,,

```
BEGIN {
c=0;
}
{
If ($1= "d")
{
c++;
printf("%s\t%s\n",$5,$11);
}
}
/*immediately after END should open braces „{,,
END{
printf("The number of packets dropped =%d\n",c);
}
```

Steps for execution

1. Open vi editor and type program. Program name should have the extension — .tcl |
[root@localhost ~]# vi lab1.tcl
2. Save the program by pressing “ESC key” first, followed by “Shift and :” keys simultaneously and type “wq” and press **Enter key**.
3. Open vi editor and type awk program. Program name should have the extension—.awk |
[root@localhost ~]# vi lab1.awk
4. Save the program by pressing “ESC key” first, followed by “Shift and :” keys simultaneously and type “wq” and press **Enter key**.
5. Run the simulation program


```
[root@localhost~]# ns lab1.tcl
```

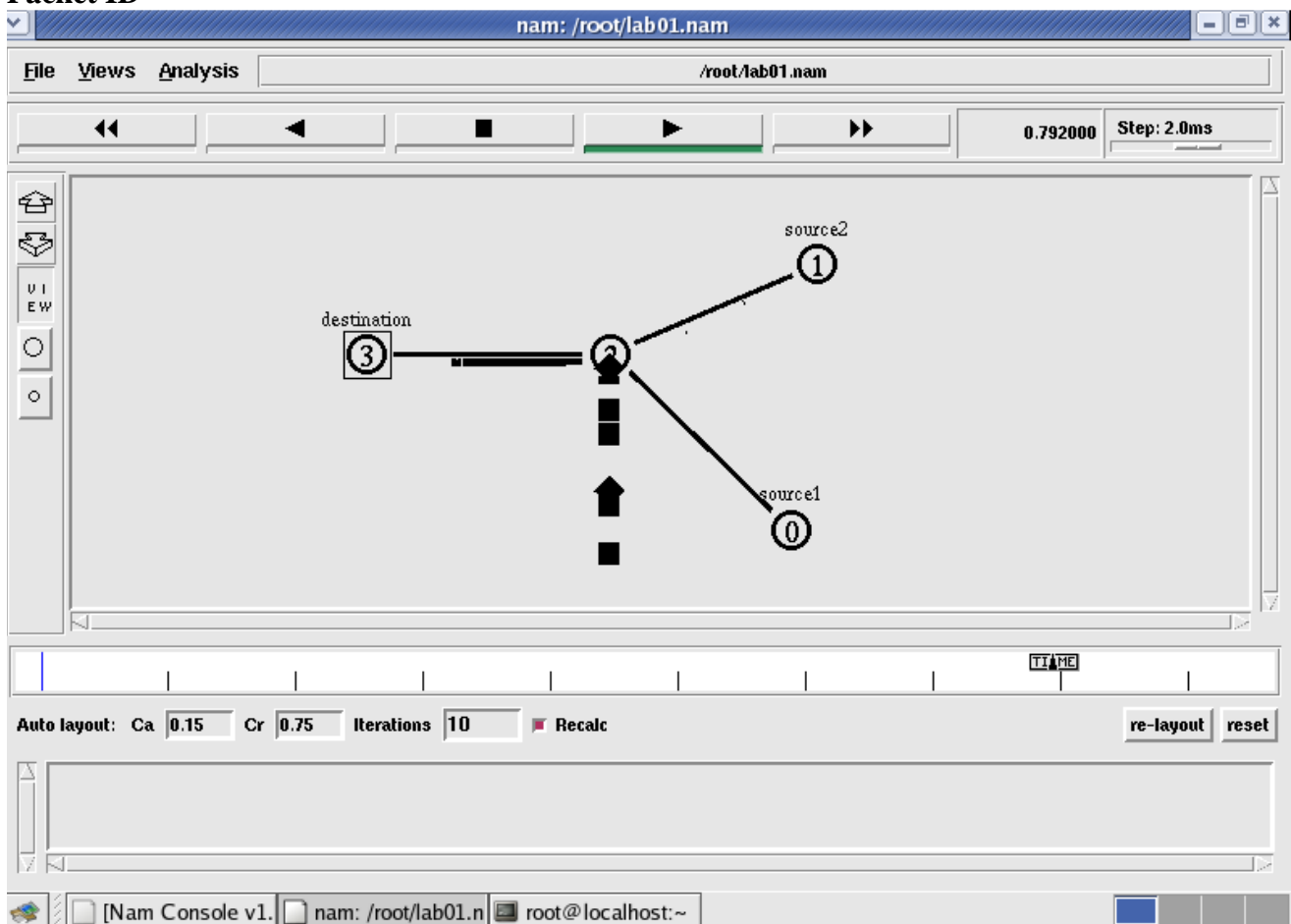
- i. Here “ns” indicates network simulator. We get the topology shown in the snapshot.
 - ii. Now press the play button in the simulation window and the simulation will begin.
6. After simulation is completed run **awk file** to see the output ,

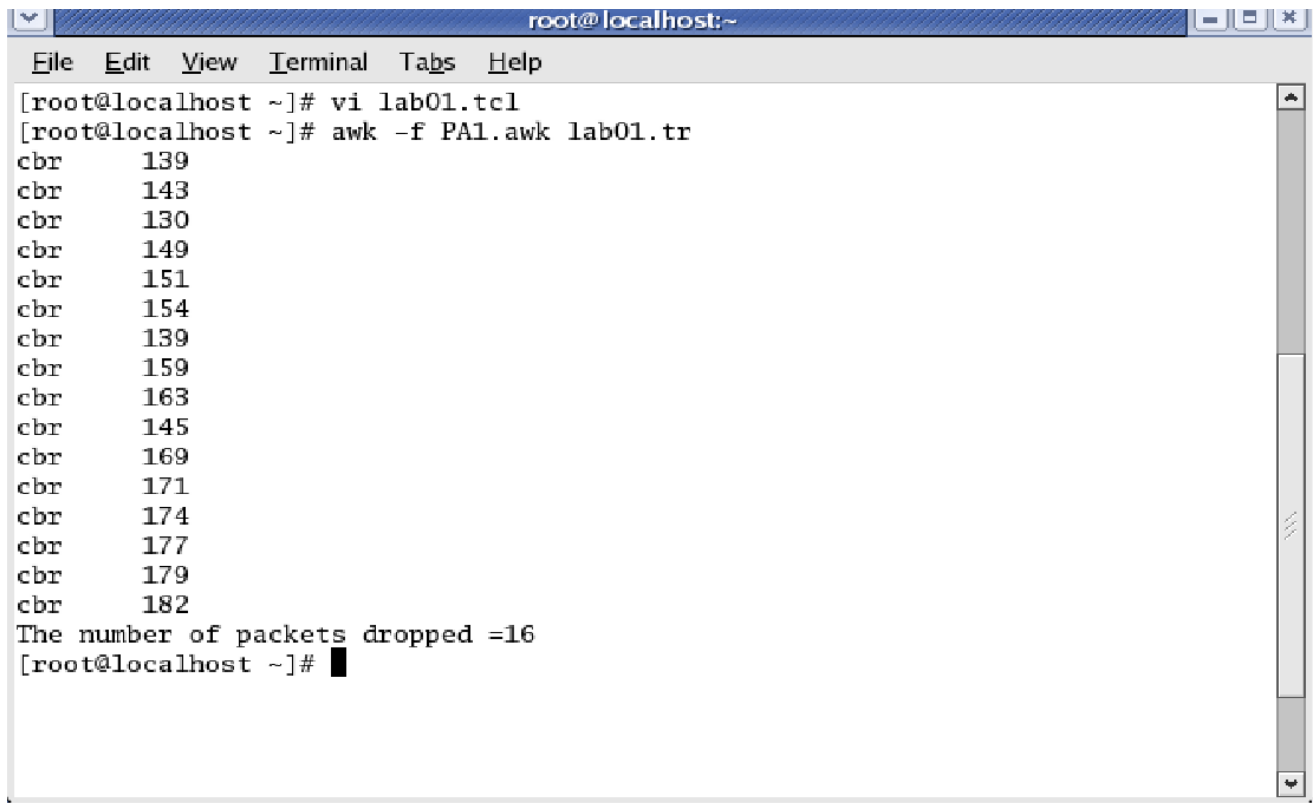
```
[root@localhost~]# awk -f lab1.awk lab1.tr
```
7. To see the trace file contents open the file as ,

```
[root@localhost~]# vi lab1.tr
```

Trace file contains 12 columns:-

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





```
root@localhost:~  
File Edit View Terminal Tabs Help  
[root@localhost ~]# vi lab01.tcl  
[root@localhost ~]# awk -f PA1.awk lab01.tr  
cbr      139  
cbr      143  
cbr      130  
cbr      149  
cbr      151  
cbr      154  
cbr      139  
cbr      159  
cbr      163  
cbr      145  
cbr      169  
cbr      171  
cbr      174  
cbr      177  
cbr      179  
cbr      182  
The number of packets dropped =16  
[root@localhost ~]#
```

Packet Drop :

I. no Bandwidth Queue limit No. of Packets dropped

1. 200Mb, 100Mb, 1Mb 10, 10 16
2. 200000b, 100Mb, 1Mb 20, 10 39
3. 200Mb, 1b, 1Mb 10000, 10 56

Note: Iterate the procedure by varying bandwidth and queue limit

Note:

1. Set the queue size fixed from n0 to n2 as 10, n1-n2 to 10 and from n2-n3 as 5.

Syntax: To set the queue size

\$ns set queue-limit <from> <to> <size> Eg:

\$ns set queue-limit \$n0 \$n2 10

2. Go on varying the bandwidth from 10, 20 30 . . and find the number of packets dropped at the node 2

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.

- PING: Packet Internet Gropper
- It is a tool used to test whether a particular host is reachable across an IP network
- It measures the time it takes for packets to be sent from the local host to a destination computer and back
- It measures and records the RRT of the packet and any issues along the way.



```
set ns [ new Simulator ]
```

```
set nf [ open lab2.nam w ]
```

```
$ns namtrace-all $nf
```

```
set tf [ open lab2.tr w ]
```

```
$ns trace-all $tf
```

```
set n0 [$ns node]
```

```
set n1 [$ns node]
```

```
set n2 [$ns node]
```

```
set n3 [$ns node]
```

```
set n4 [$ns node]
```

```
set n5 [$ns node]
```

```
$n4 shape box
```

```
$ns duplex-link $n0 $n4 1005Mb 1ms DropTail
```

```
$ns duplex-link $n1 $n4 50Mb 1ms DropTail
```

```
$ns duplex-link $n2 $n4 2000Mb 1ms DropTail
```

```
$ns duplex-link $n3 $n4 200Mb 1ms DropTail
```

```
$ns duplex-link $n4 $n5 1Mb 1ms DropTail
```

```
set p1 [new Agent/Ping]
```

```
$ns attach-agent $n0 $p1
```

```
$p1 set packetSize_ 50000
```

```
$p1 set interval_ 0.0001
```

```
set p2 [new Agent/Ping]
```

```
$ns attach-agent $n1 $p2
```

```
set p3 [new Agent/Ping]
```

```
$ns attach-agent $n2 $p3
```

```
$p3 set packetSize_ 30000
$p3 set interval_ 0.00001

set p4 [new Agent/Ping]
$ns attach-agent $n3 $p4

set p5 [new Agent/Ping]
$ns attach-agent $n5 $p5

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

Agent/Ping instproc recv {from rtt} {
$self instvar node_
puts "node [$node_ id] received answer from $from with round trip time $rtt msec"
}
# please provide space between $node_ and id. No space between $ and from. No
#space between and $ and rtt */
$ns connect $p1 $p5
$ns connect $p3 $p4

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

$ns at 0.1 "$p1 send"
$ns at 0.2 "$p1 send"
$ns at 0.3 "$p1 send"
$ns at 0.4 "$p1 send"
$ns at 0.5 "$p1 send"
$ns at 0.6 "$p1 send"
$ns at 0.7 "$p1 send"
$ns at 0.8 "$p1 send"
$ns at 0.9 "$p1 send"
$ns at 1.0 "$p1 send"
$ns at 1.1 "$p1 send"
$ns at 1.2 "$p1 send"
$ns at 1.3 "$p1 send"
$ns at 1.4 "$p1 send"
```

\$ns at 1.5 "\$p1 send"
\$ns at 1.6 "\$p1 send"
\$ns at 1.7 "\$p1 send"
\$ns at 1.8 "\$p1 send"
\$ns at 1.9 "\$p1 send"
\$ns at 2.0 "\$p1 send"
\$ns at 2.1 "\$p1 send"
\$ns at 2.2 "\$p1 send"
\$ns at 2.3 "\$p1 send"
\$ns at 2.4 "\$p1 send"
\$ns at 2.5 "\$p1 send"
\$ns at 2.6 "\$p1 send"
\$ns at 2.7 "\$p1 send"
\$ns at 2.8 "\$p1 send"
\$ns at 2.9 "\$p1 send"

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

\$ns run

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

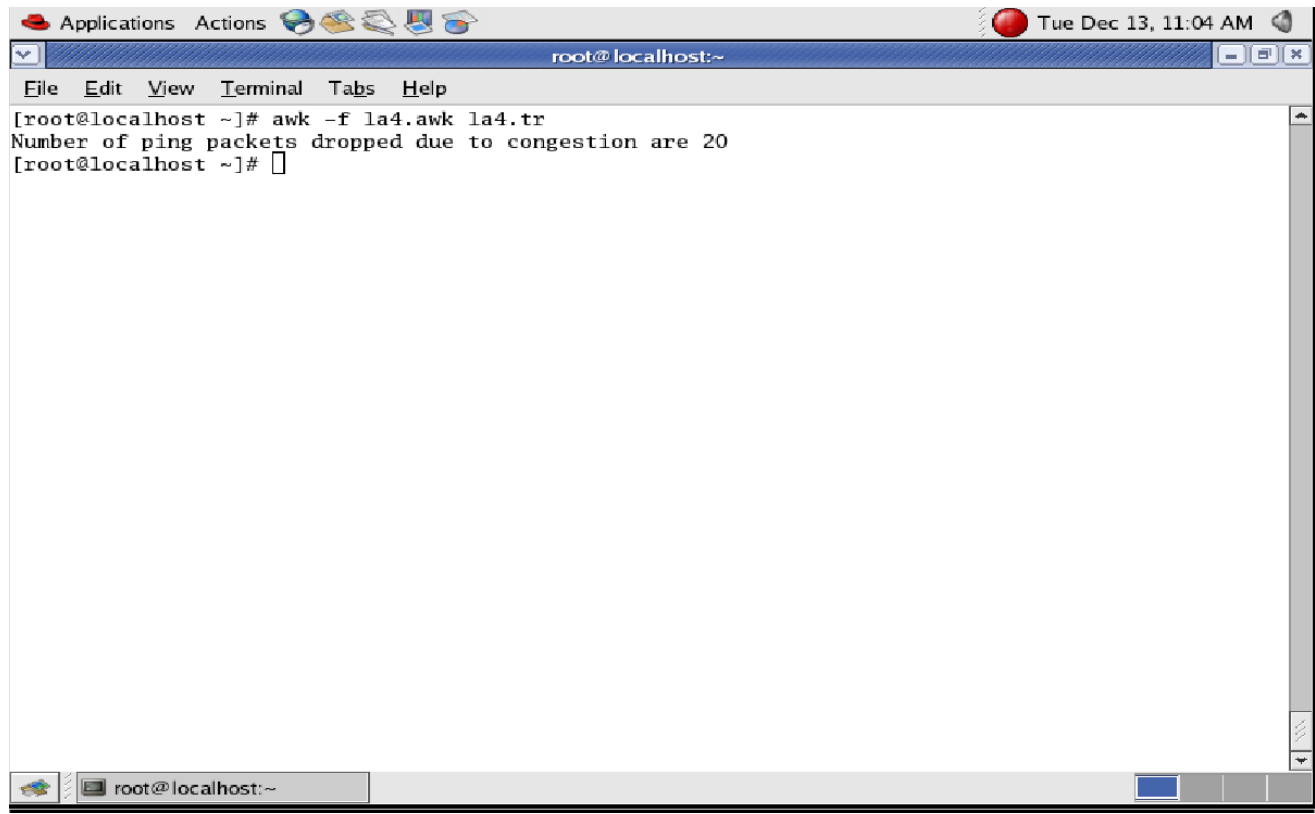
```
BEGIN{
drop=0;
}
{
if($1=="d" )
{
drop++;
}
}
END{
printf("Total number of %s packets dropped due to congestion =%d\n",$5,drop);
}
```

Steps for execution

1. Open vi editor and type program. Program name should have the extension — **.tcl** ||
[root@localhost ~]# vi lab2.tcl
2. Save the program by pressing “**ESC key**” first, followed by “**Shift and :**” keys simultaneously and type “**wq**” and press **Enter key**.
3. Open vi editor and type **awk** program. Program name should have the extension — **.awk** ||
4. **[root@localhost ~]# vi lab2.awk**
 - 1) Save the program by pressing “**ESC key**” first, followed by “**Shift and :**” keys simultaneously and type “**wq**” and press **Enter key**.
 - 2) Run the simulation program
6. **[root@localhost~]# ns lab2.tcl**
 - a. Here “**ns**” indicates network simulator. We get the topology shown in the snapshot.
 - b. Now press the play button in the simulation window and the simulation will begins.
7. After simulation is completed run **awk file** to see the output ,**[root@localhost~]# awk -f lab2.awk lab2.tr**
8. To see the trace file contents open the file as ,
[root@localhost~]# vi lab2.tr

Topology





The screenshot shows a terminal window titled "root@localhost:~". The window has a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal content shows the command `awk -f la4.awk la4.tr` being executed, which outputs "Number of ping packets dropped due to congestion are 20". The prompt `[root@localhost ~]#` is visible at the end of the line. The window's title bar includes "Applications", "Actions", and a clock showing "Tue Dec 13, 11:04 AM".

```
[root@localhost ~]# awk -f la4.awk la4.tr
Number of ping packets dropped due to congestion are 20
[root@localhost ~]#
```

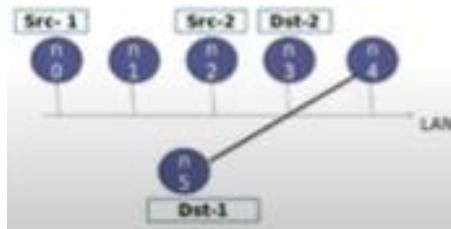
Note:

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

Program 3

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

- local Area Network (LAN) is a data communication network connecting various terminals or computers within a building or limited geographical area.
- Ethernet is the most widely used LAN technology.
- Ethernet generally uses Bus Topology.
- Ethernet operates in two layers of the OSI model, Physical Layer, and Data Link Layer.
- It is having 2 sublayers: Medium access (Mac) Layer: 802.3 & Logical link layer



```
set ns [new Simulator]
```

```
set tf [open lab3.tr w]
```

```
$ns trace-all $tf
```

```
set nf [open lab3.nam w]
```

```
$ns namtrace-all $nf
```

```
set n0 [$ns node]
```

```
$n0 color "magenta"
```

```
$n0 label "src1"
```

```
set n1 [$ns node]
```

```
set n2 [$ns node]
```

```
$n2 color "magenta"
```

```
$n2 label "src2"
```

```
set n3 [$ns node]
```

```
$n3 color "blue"
```

```
$n3 label "dest2"
```

```
set n4 [$ns node]
```

```
set n5 [$ns node]
```

```
$n5 color "blue"
```

```
$n5 label "dest1"
```

```
$ns make-lan "$n0 $n1 $n2 $n3 $n4" 100Mb 100ms LL Queue/DropTail Mac/802_3
```

```
/* should come in single line */
```

```
$ns duplex-link $n4 $n5 1Mb 1ms DropTail
```

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

```
set sink5 [new Agent/TCPSink]
$ns attach-agent $n5 $sink5
$ns connect $tcp0 $sink5
```

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

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

```
$ns connect $tcp2 $sink3
```

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

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

```
proc finish { } {
    global ns 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
```

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

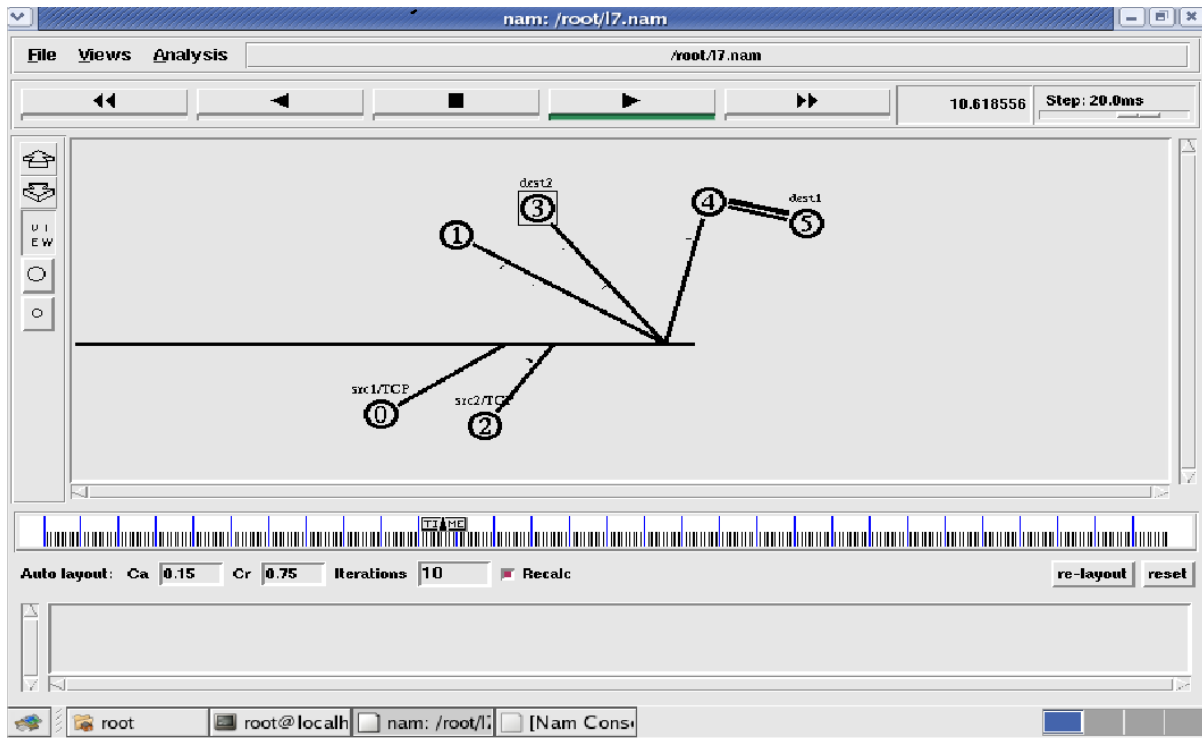
cwnd:- means congestion window

```
BEGIN {
}
{
if($6=="cwnd_") /* don't leave space after writing cwnd_ */
printf("%f\t%f\t\n",$1,$7); /* you must put \n in printf */
}
END {
}
```

Steps for execution

1. Open vi editor and type program. Program name should have the extension — **.tcl** |
[root@localhost ~]# vi lab3.tcl
Save the program by pressing “ESC key” first, followed by “Shift and :” keys simultaneously and type “wq” and press **Enter key**.
2. Open vi editor and type **awk** program. Program name should have the extension—**.awk** |
3. **[root@localhost ~]# vi lab3.awk**
Save the program by pressing “ESC key” first, followed by “Shift and :” keys simultaneously and type “wq” and press **Enter key**.
4. Run the simulation program
[root@localhost~]# ns lab3.tcl
5. After simulation is completed run **awk file** to see the output ,
 - i. **[root@localhost~]# awk -f lab3.awk file1.tr > a1**
 - ii. **[root@localhost~]# awk -f lab3.awk file2.tr > a2**
 - iii. **[root@localhost~]# xgraph a1 a2**
6. Here we are using the congestion window trace files i.e. **file1.tr** and **file2.tr** and we are redirecting the contents of those files to new files say **a1** and **a2** using **output redirection operator (>)**.
7. To see the trace file contents open the file as **[root@localhost~]# vi lab3.tr**
- 8.

Topology



Output

