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 */
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**.

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.

```
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)

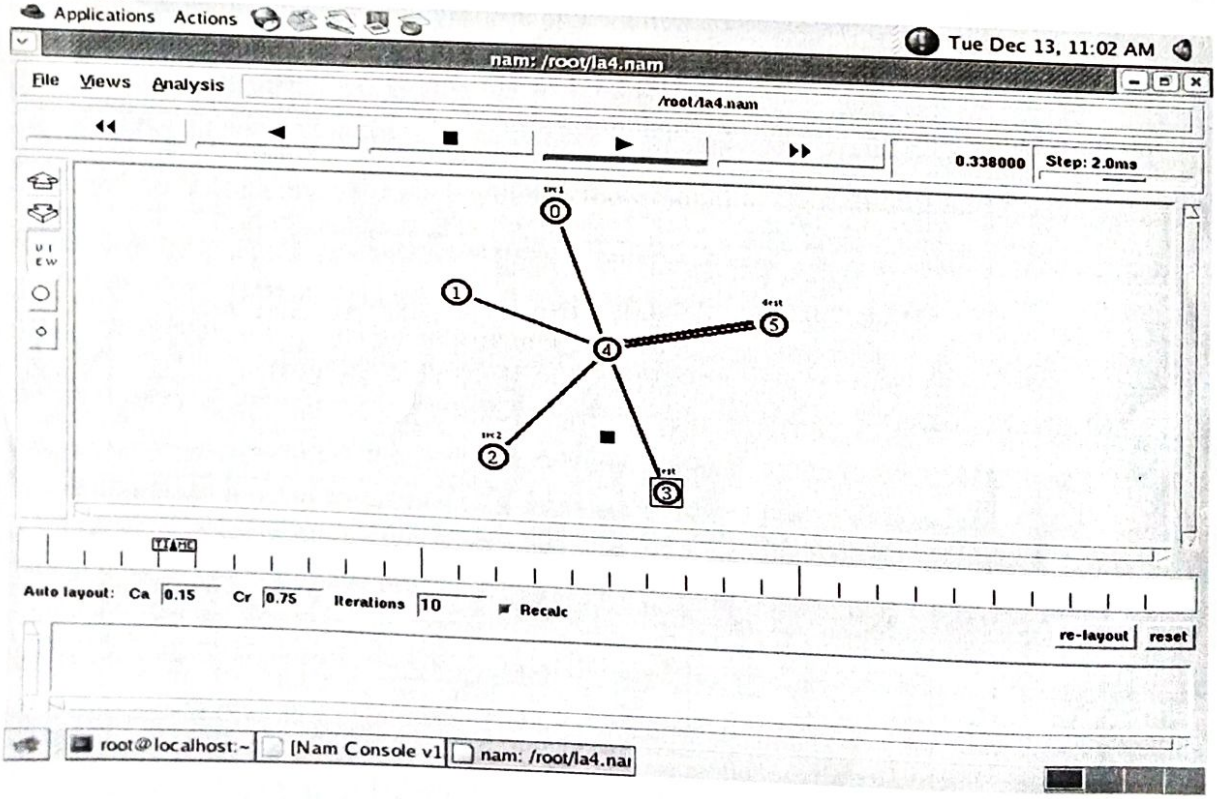
```
BEGIN{
drop=0;
}
{
 if($1= ="d" )
  {
   drop++;
  }
```

```
} 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** "
   [root@localhost ~]# vi lab2.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 lab2.tcl
   i) Here **"ns"** indicates network simulator. We get the topology shown in the snapshot.
   ii) Now press the play button in the simulation window and the simulation will begins.
6) After simulation is completed run **awk file** to see the output ,
   [root@localhost~]# awk –f lab2.awk lab2.tr
7) To see the trace file contents open the file as ,
   [root@localhost~]# vi lab2.tr

Topology



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

- 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.

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

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

International Standard CRC Polynomials

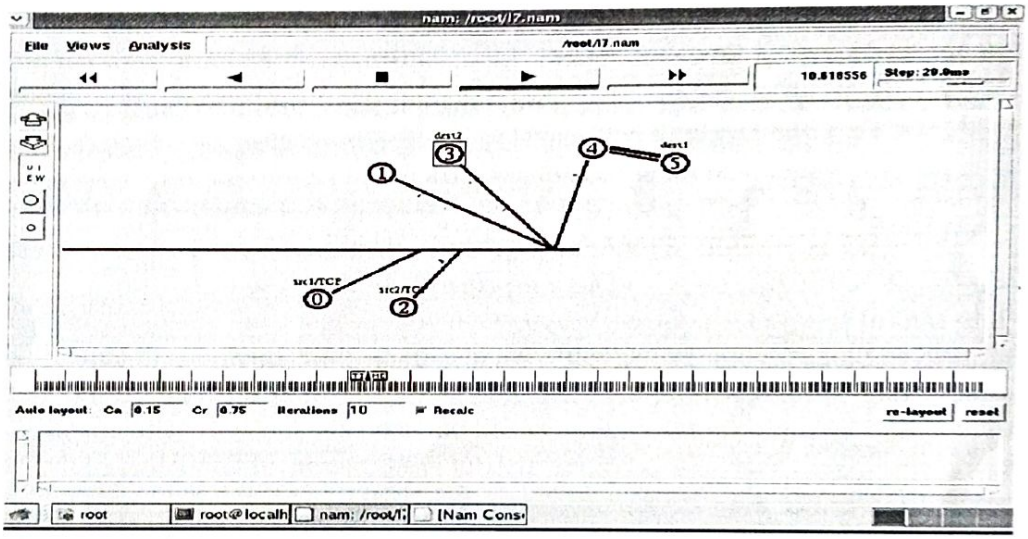
4) Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.
5) Run the simulation program

[root@localhost~]# ns lab3.tcl

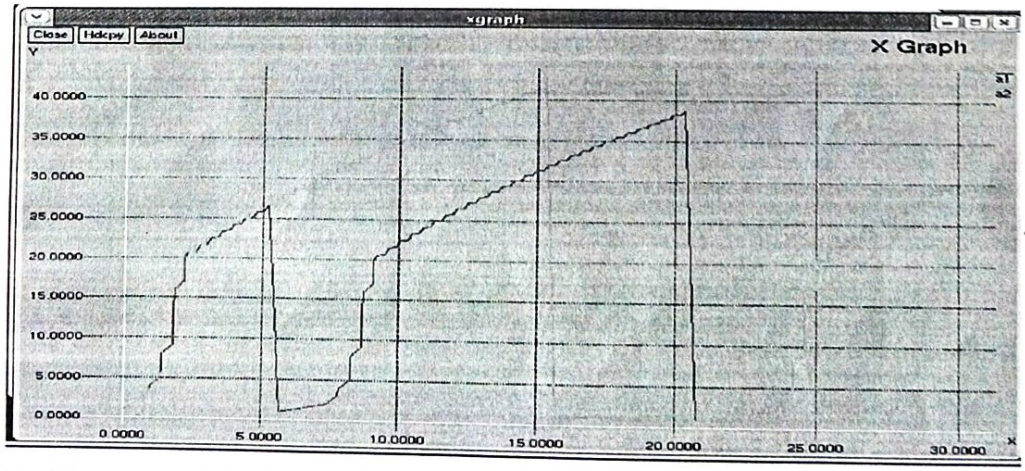
6) After simulation is completed run **awk file** to see the output ,
   i. **[root@localhost~]# awk –f lab3.awk file1.tr > a1**
   ii. **[root@localhost~]# awk –f lab3.awk file2.tr > a2**
   iii. **[root@localhost~]# xgraph a1 a2**
7) Here we are using the congestion window trace files i.e. **file1.tr** and **file2.tr** and we are redirecting the contents of those files to new files say **a1** and **a2** using **output redirection operator (>)**.
8) To see the trace file contents open the file as ,

[root@localhost~]# vi lab3.tr

Topology



Output



Source Code:

```
/******************************************************************************
 *  Compilation:  javac CRC16CCITT.java
 *  Execution:    java CRC16CCITT s
 *  Dependencies:
 *
 *  Reads in a sequence of bytes and prints out its 16 bit
 *  Cylcic Redundancy Check (CRC-CCIIT 0xFFFF).
 *
 *  1 + x + x^5 + x^12 + x^16 is irreducible polynomial.
 *
 *  % java CRC16-CCITT 123456789
 *  CRC16-CCITT = 29b1
 *
 ******************************************************************************/
public class CRC16CCITT {
   public static void main(String[] args) {
      int crc = 0xFFFF;          // initial value
      int polynomial = 0x1021;   // 0001 0000 0010 0001  (0, 5, 12)

      // byte[] testBytes = "123456789".getBytes("ASCII");

      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;
      StdOut.println("CRC16-CCITT = " + Integer.toHexString(crc));
   }

}
```

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

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

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

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

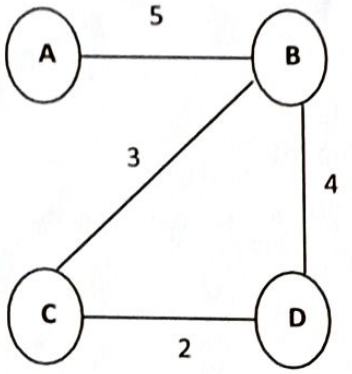
Source code:

```
import java.util.Scanner;

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

        public BellmanFord(int num_ver)
        {
                this.num_ver = num_ver; D =
                new int[num_ver + 1];
        }

        public void BellmanFordEvaluation(int source, int A[][])
        {
                for (int node = 1; node <= num_ver; node++)
                {
                        D[node] = MAX_VALUE;

                }
                D[source] = 0;
                for (int node = 1; node <= num_ver - 1; node++)
```

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

public static void main(String[ ] args)
{
      int num_ver =
      0; int source;
      Scanner scanner = new Scanner(System.in);
      System.out.println("Enter the number of
      vertices"); num_ver = scanner.nextInt();
      int A[][] = new int[num_ver + 1][num_ver + 1];
     System.out.println("Enter the adjacency matrix");
     for (int sn = 1; sn <= num_ver; sn++)
     {
             for (int dn = 1; dn <= num_ver; dn++)
             {


                 A[sn][dn] = scanner.nextInt(); if
                  (sn == dn)
                 {
```

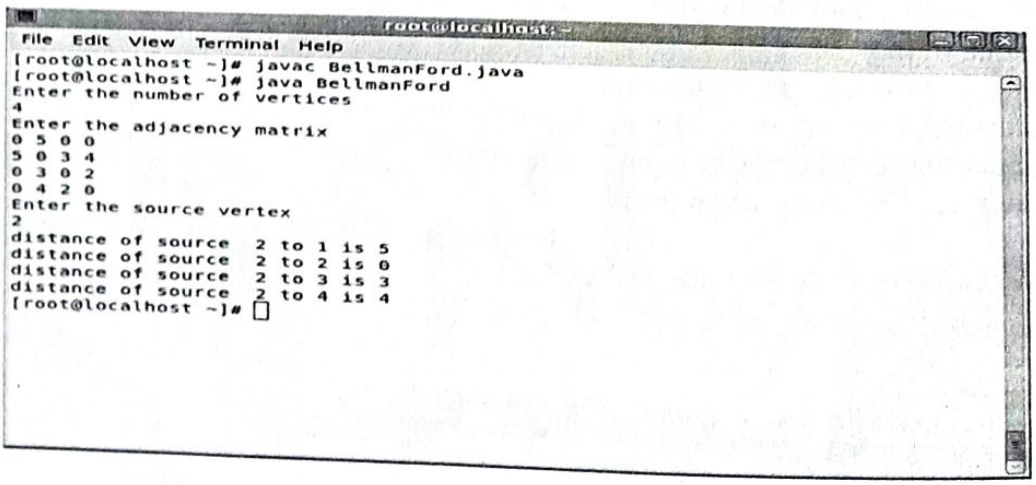
```
                        A[sn][dn] = 0;
                        continue;
                    }
                    if (A[sn][dn] == 0)
                    {
                        A[sn][dn] = MAX_VALUE;
                    }
                }
            }
            System.out.println("Enter the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

Input graph:



Output:

[root@localhost ~]# vi BellmanFord.java



6. Using TCP/IP sockets, write a client – server program to make the client send the file name and to make the server send back the contents of the requested file if present. Implement the above program using as message queues or FIFOs as IPC channels.

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

Source Code:

TCP Client

```
import java.io.*;
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());
out = new DataOutputStream(socket.getOutputStream());
        out.writeUTF(filename);
        String fileContent = in.readUTF();

        if (fileContent.length() > 0)
            System.out.println(fileContent);
        else
            System.out.println("FILE IS EMPTY");
    }
}
```

TCP Server

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

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

parameters, methods description in socket communication.

Output:

At server side:

```
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");
                }
        }
}
```

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

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

Source Code:

UDP Client

```
import java.net.*;
import java.util.Scanner;
public class DSender
{
  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("\nEnter 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);
        }
  }
}
```

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

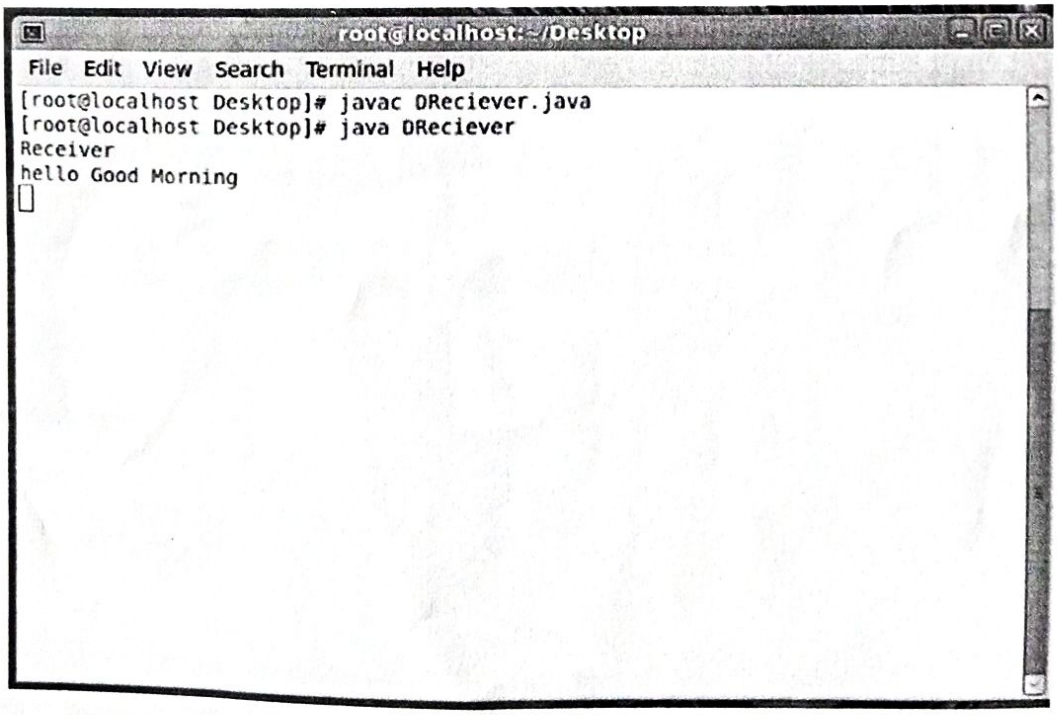
1) Open a terminal run the server program.
2) Open one more terminal run the client program, the sent message will be received.

UDP Server

```
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");
        }
    }
}
```

At Server side:



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

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

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

Key Generation Algorithm

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

Encryption

Sender A does the following:-

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

Decryption

Recipient B does the following:-

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

Source Code:

Key Generation

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

class RSA
{
      public static void main(String[] args)
      {
```

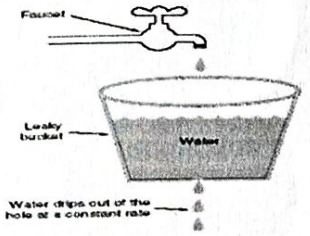
```
        Random r = new Random();
        Random r2 = new Random();
        int e = Integer.parseInt(args[0]);
        BigInteger p = BigInteger.probablePrime(4,r);
        BigInteger q = BigInteger.probablePrime(4,r2);
        BigInteger n = p.multiply(q);
        BigInteger p1 = p.subtract(new BigInteger("1"));
        BigInteger q1 = q.subtract(new BigInteger("1"));
        BigInteger phi = p1.multiply(q1);
        while(true)
        {
            BigInteger GCD = phi.gcd(new BigInteger(""+e));

            if(GCD.equals(BigInteger.ONE))
            {
                break;
            }
            e++;
        }
        BigInteger a = new BigInteger(""+e);
        BigInteger d = a.modInverse(phi);
        System.out.println("public key:"+a+","+n);
        System.out.println("Detection key:"+d+","+n);
    }
}
```

Encryption and Decryption

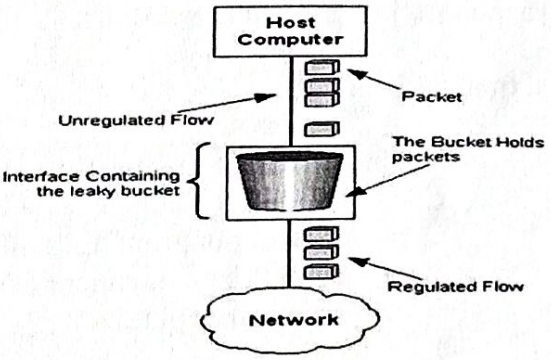
```
import java.util.*;
import java.math.BigInteger;
import java.lang.*;

class ed {
public static void main(String[] args) {

    BigInteger pubKey = new BigInteger(args[0]);
      BigInteger prvKey = new BigInteger(args[1]);
    BigInteger n = new BigInteger(args[2]);
      int asciiVal = Integer.parseInt(args[3]);
      BigInteger val = new BigInteger(""+ asciiVal);
      BigInteger cKey = val.modPow(pubKey,n);

    System.out.println("Cipher Text: " + cKey);
    BigInteger pVal = cKey.modPow(prvKey,n);

    int Pval = pVal.intValue();
    System.out.println("Plain text: " + Pval);
    }
}
```

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

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



(a) A leaky bucket with water



(b) A leaky bucket with packets

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

Source Code:

```
import java.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");
```

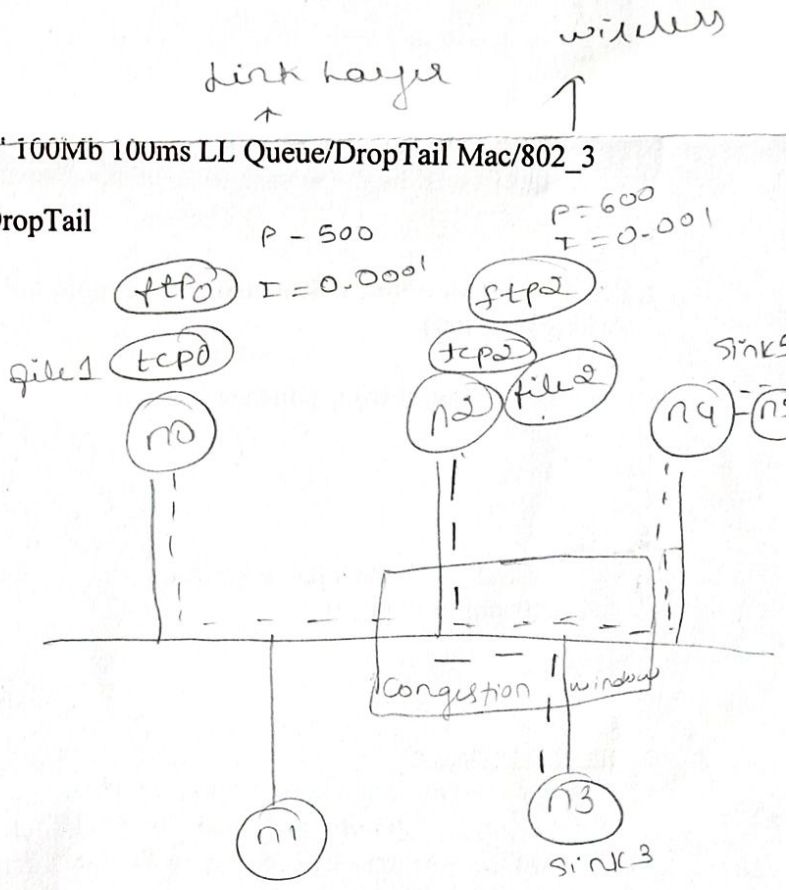
```
        }

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");
        }
    }
```

Output:

```
File  Edit  View  Terminal  Help
Enter the packets to be sent:
12
Enter 0 element: 2
Enter 1 element: 3
Enter 2 element: 5
Enter 3 element: 6
Enter 4 element: 8
Enter 5 element: 9
Enter 6 element: 4
Enter 7 element: 5
Enter 8 element: 6
Enter 9 element: 2
Enter 10 element: 3
Queue is full
Lost Packet: 3
```

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

```
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
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 lab3.awk