**EX.NO 1(a) Implementation of Sliding Window Protocol**

**Date**

**AIM**

To write a java program to implement sliding window protocol.

**ALGORITHM**

1.Start the program.

2.Get the number of frames from the user

3.Create the frame based on the user request.

4.Send frames to serverfrom the client side.

5.If frame reach the server it will send ACK signal to client otherwise it will send NACK signal toclient.

6.Stop the program.

**PROGRAM**

**SENDER PROGRAM**

import java.net.\*;

import java.io.\*;

importjava.rmi.\*;

public class SlidSender

{

public static void main(String a[])throws Exception

{

ServerSocket ser=new ServerSocket(10);

Socket s=ser.accept();

DataInputStream in=new DataInputStream(System.in);

DataInputStream in1=newDataInputStream(s.getInputStream());

String sbuff[]=new String[8];

PrintStream p;

int sptr=0,sws=8,nf,ano,i;

String ch;

do

{

p=new PrintStream(s.getOutputStream());

System.out.print("Enter theno. of frames : ");

nf=Integer.parseInt(in.readLine());

p.println(nf);

if(nf<=sws-1)

{

System.out.println("Enter "+nf+" Messages to be send\n");

for(i=1;i<=nf;i++)

{

sbuff[sptr]=in.readLine();

p.println(sbuff[sptr]);

sptr=++sptr%8;

}

sws-=nf;

System.out.print("Acknowledgment received"); ano=Integer.parseInt(in1.readLine());

System.out.println(" for "+ano+" frames");

sws+=nf;

}

else

{

System.out.println("The no. of frames exceeds window size");

break;

}

System.out.print("\nDo you wants to send some more frames : "); ch=in.readLine();

p.println(ch);

}

while(ch.equals("yes"));

s.close();

}

}

**RECEIVER PROGRAM**

import java.net.\*;

import java.io.\*;

class SlidReceiver

{

public static void main(String a[])throws Exception

{

Socket s=new Socket(InetAddress.getLocalHost(),10);

DataInputStream in=newDataInputStream(s.getInputStream());

PrintStream p=new PrintStream(s.getOutputStream());

int i=0,rptr=-1,nf,rws=8;

String rbuf[]=new String[8];

String ch;

System.out.println();

do

{

nf=Integer.parseInt(in.readLine());

if(nf<=rws-1)

{

for(i=1;i<=nf;i++)

{

rptr=++rptr%8;

rbuf[rptr]=in.readLine();

System.out.println("The received Frame " +rptr+" is : "+rbuf[rptr]);

}

rws-=nf;

System.out.println("\nAcknowledgmentsent\n");

p.println(rptr+1);

rws+=nf;

}

else

break;

ch=in.readLine();

}

while(ch.equals("yes"));

}

}

**OUTPUT**

**OUTPUT FOR SENDER**

Enter the no. of frames : 4

Enter 4 Messages to be send

hi

how r u

i am fine

how is evryone

Acknowledgment received for 4 frames

Do you wants to send some more frames : no

**OUTPUT FOR RECEIVER**

The received Frame 0 is : hi

The received Frame 1 is : how r u

The received Frame 2 is : i am fine

The received Frame 3 is : how is everyone

**RESULT**

Thus the implementation of Sliding Window Protocol is done successfully.

**EX.NO 1(b) Implementation of Stop and Wait Protocol**

**Date**

**AIM**

To write a java program to implementstop and wait protocol.

**ALGORITHM**

1.Start the program.

2.Get the number of frames from the user

3.Create the frame based on the user request.

4.Send framesto server from the client side one by one after receiving the ACK for each frame.

5.If frame reach the server it will send ACK signal to client otherwise it will send NACKsignal to client.

6.Stop the program.

**PROGRAM**

**SENDER PROGRAM**

import java.io.\*;

import java.net.\*;

import java.util.Scanner;

class StopWaitSender

{

public static void main(String args[]) throws Exception

{

StopWaitSender sws = new StopWaitSender();

sws.run();

}

public void run() throws Exception

{

Scanner sc=new Scanner(System.in);

System.out.println(“Enter no of frames to be sent:”);

int n=sc.nextInt();

Socket myskt=new Socket(“localhost”,9999);

PrintStream myps=new PrintStream(myskt.getOutputStream());

for(int i=0;i<=n;)

{

if(i==n)

{

myps.println(“exit”);

break;

}

System.out.println(“Frame no “+i+” is sent”);

myps.println(i);

BufferedReader bf=new BufferedReader(new InputStreamReader(myskt.getInputStream()));

String ack=bf.readLine();

if(ack!=null)

{

System.out.println(“Acknowledgement was Received from receiver”);

i++;

Thread.sleep(4000);

}

else

{

myps.println(i);

}

}

}

}

**RECEIVER PROGRAM**

import java.io.\*;

import java.net.\*;

class StopWaitReceiver

{

public static void main(String args[])throws Exception

{

StopWaitReceiver swr = new StopWaitReceiver();

swr.run();

}

public void run() throws Exception

{

String temp=”any message”,str=”exit”;

ServerSocket myss=new ServerSocket(9999);

Socket ss\_accept=myss.accept();

BufferedReader ss\_bf=new BufferedReader(new InputStreamReader(ss\_accept.getInputStream()));

PrintStream myps=new PrintStream(ss\_accept.getOutputStream());

while(temp.compareTo(str)!=0)

{

Thread.sleep(1000);

temp=ss\_bf.readLine();

if(temp.compareTo(str)==0)

{ break;}

System.out.println(“Frame “+temp+” was received”);

Thread.sleep(500);

myps.println(“Received”);

}

System.out.println(“ALL FRAMES WERE RECEIVED SUCCESSFULLY”);

}

}

**OUTPUT**

**OUTPUT FOR SENDER**

C:\javaprog>javac StopWaitSender.java

C:\javaprog>java StopWaitSender

Enter no of frames to be sent:

4

Frame no 0 is sent

Acknowledgement was Received from receiver

Frame no 1 is sent

Acknowledgement was Received from receiver

Frame no 2 is sent

Acknowledgement was Received from receiver

Frame no 3 is sent

Acknowledgement was Received from receiver

**OUTPUT FOR RECEIVER**

C:\javaprog>javac StopWaitReceiver.java

C:\javaprog>java StopWaitReceiver

Frame 0 was received

Frame 1 was received

Frame 2 was received

Frame 3 was received

ALL FRAMES WERE RECEIVED SUCCESSFULLY

**RESULT**

Thus the implementation of Sliding Stop and Wait Protocol is done successfully.

**EX.NO 2 Implementation of Client** – **Server model**

**Date**

**AIM**

To implement a socket program to display server’s date and time in client using TCP Socket.

**ALGORITHM**

**Server**

1. Create a server socket and bind it to port.

2. Listen for new connection and when a connection arrives, accept it.

3. Send servers date and time to the client.

4. Read clients IP address sent by the client.

5. Display the client details.

6. Stop.

**Client**

1. Create a client socket and connect it to the server‟s port number.

2. Retrieve its own IP address using built-in function.

3. Send its IP address to the server.

4. Display the date & time sent by the server.

5. Stop.

**PROGRAM**

**DateClient.java**

import java.net.\*;

import java.io.\*;

class DateClient

{

public static void main (String args[])

{

Socket soc;

DataInputStream dis;

String sdate;

PrintStream ps;

try

{

InetAddress ia=InetAddress.getLocalHost();

soc=new Socket(ia,8020);

dis=new DataInputStream(soc.getInputStream());

sdate=dis.readLine();

System.out.println("THE date in the server is:"+sdate);

ps=new PrintStream(soc.getOutputStream());

ps.println(ia);

}

catch(IOException e)

{

System.out.println("THE EXCEPTION is :"+e);

}

}

}

**DateServer.java**

import java.net.\*;

import java.io.\*;

import java.util.\*;

class DateServer

{

public static void main(String args[])

{

ServerSocket ss;

Socket s;

PrintStream ps;

DataInputStream dis;

String inet;

try

{

ss=new ServerSocket(8020);

while(true)

{

s=ss.accept();

ps=new PrintStream(s.getOutputStream());

Date d=new Date();

ps.println(d);

dis=new DataInputStream(s.getInputStream());

inet=dis.readLine();

System.out.println("THE CLIENT SYSTEM ADDRESS IS :"+inet);

ps.close();

}

}

catch(IOException e)

{

System.out.println("The exception is :"+e);

}

}

}

**OUTPUT**

**OUTPUT FOR CLIENT**

C:\Program Files\Java\jdk1.5.0\bin>javac DateClient.java

C:\Program Files\Java\jdk1.5.0\bin>java DateClient

THE date in the server is: Wed Oct03 10:13:58IST 2018

**OUTPUT FOR SERVER**

C:\Program Files\Java\jdk1.5.0\bin>javac DateServer.java

C:\Program Files\Java\jdk1.5.0\bin>java DateServer

THE CLIENT SYSTEM ADDRESS IS :2-9-33/192.168.6.33

**RESULT**

Thus the program to display server’s date and time in client using TCP Socket is implemented successfully.

**EX.NO 3(a) Implementation of Address Resolution Protocol (ARP)**

**Date**

**AIM**

To write a java program for simulating ARP protocol.

**ALGORITHM**

**Client**

1. Start the program

2. Using Socket, connection is established between client and server.

3. Get the IP address to be converted into MAC address.

4. Send this IP address to server.

5**.** Read MAC from server and print in the console.

**Server**

1. Start the program

2. Accept the socket which is created by the client.

3. Server maintains the table in which IP and corresponding MAC addresses are stored.

4. Read the IP address which is sent by the client.

5. Map the IP address with its MAC address and return the MAC address to client.

**PROGRAM**

**Client**

import java.io.\*;

import java.net.\*;

import java.util.\*;

class ClientARP

{

public static void main(String args[])

{

try

{

BufferedReader in=new BufferedReader(new InputStreamReader(System.in));

Socket clsct=new Socket("127.0.0.1",8080);

DataInputStream din=new DataInputStream(clsct.getInputStream());

DataOutputStream dout=new DataOutputStream(clsct.getOutputStream());

System.out.println("Enter the Logical address(IP):");

String str1=in.readLine();

dout.writeBytes(str1+'\n');

String str=din.readLine();

System.out.println("The Physical Address is: "+str);

clsct.close();

}

catch (Exception e)

{

System.out.println(e);

}

}

}

**Server**

import java.io.\*;

import java.net.\*;

import java.util.\*;

class ServerARP

{

public static void main(String args[])

{

try

{

ServerSocket obj=new ServerSocket(8080);

Socket obj1=obj.accept();

while(true)

{

DataInputStream din=new DataInputStream(obj1.getInputStream());

DataOutputStream dout=new DataOutputStream(obj1.getOutputStream());

String str=din.readLine();

String ip[]={"192.168.6.33","192.168.6.34"};

String mac[]={"6A:08:AA:C2","8A:BC:E3:FA"};

for(int i=0;i<ip.length;i++)

{

if(str.equals(ip[i]))

{

dout.writeBytes(mac[i]+'\n');

break;

}

}

obj.close();

}

}

catch(Exception e)

{

System.out.println(e);

}

}

}

**OUTPUT**

Enter the Logical address(IP):

192.168.6.33

The Physical Address is: 6A:08:AA:C2

**Result**:

Thus the program to simulate ARP protocols is implemented successfully.

**EX.NO3(b) Implementation of Reverse Address Resolution Protocol(RARP)**

**Date**

**AIM**

To write a java program for simulating RARP protocol.

**ALGORITHM**

**Client**

1.Start the program

2.Using Socket, connection is established between client and server.

3.Get the MAC address to be converted into IP address.

4.Send this MAC address to server.

5.Server returns the IP address to client.

**Server**

1. Start the program.

2. Server maintains the table in which IP and corresponding MAC addresses are stored.

3. Read the MAC address which is send by the client.

4. Map the IP address with its MAC address and return the IP address to client.

**PROGRAM**

**Client**

import java.io.\*;

import java.net.\*;

import java.util.\*;

class ClientRARP

{

public static void main(String args[])

{

try

{

DatagramSocket client=new DatagramSocket();

InetAddress addr=InetAddress.getByName("127.0.0.1");

byte[] sendbyte=new byte[1024];

byte[] receivebyte=new byte[1024];

BufferedReader in=new BufferedReader(new InputStreamReader(System.in));

System.out.println("Enter the Physical address (MAC):");

String str=in.readLine(); sendbyte=str.getBytes();

DatagramPacket sender=new DatagramPacket(sendbyte,sendbyte.length,addr,1309);

client.send(sender);

DatagramPacket receiver=new DatagramPacket(receivebyte,receivebyte.length);

client.receive(receiver);

String s=new String(receiver.getData());

System.out.println("The Logical Address is(IP): "+s.trim());

client.close();

}

catch(Exception e)

{

System.out.println(e);

}

}

}

**Server**

import java.io.\*;

import java.net.\*;

import java.util.\*;

class ServerRARP

{

public static void main(String args[])

{

try

{

DatagramSocket server = new DatagramSocket(1309);

while(true)

{

byte[] sendbyte=new byte[1024];

byte[] receivebyte=new byte[1024];

DatagramPacket receiver=new DatagramPacket(receivebyte,receivebyte.length);

server.receive(receiver);

String str=new String(receiver.getData());

String s=str.trim();

InetAddress addr=receiver.getAddress();

int port=receiver.getPort();

Stringip[]={"192.168.6.33","192.168.6.34"};

Stringmac[]={"6A:08:AA:C2","8A:BC:E3:FA"};

for(int i=0;i<ip.length;i++)

{

if(s.equals(mac[i]))

{

sendbyte=ip[i].getBytes();

DatagramPacket sender=new DatagramPacket(sendbyte,sendbyte.length,addr,port);

server.send(sender);

break;

}

}

break;

}

}

catch(Exception e)

{

System.out.println(e);

}

}

}

**OUTPUT**

Enter the Physical address(MAC): 6A:08:AA:C2

The Logical Address is(IP): 192.168.6.33

**RESULT**

Thus the program to simulate RARP protocols is implemented successfully.

**EX-NO 4(a) Implementation of PING Command**

**Date**

**AIM**

To write a java program for simulating PING Command.

**ALGORITHM**

Step 1: start the program.

Step 2: Include necessary package in java.

Step 3: Create a process object to implement the ping command.

Step 4: Execute the system command and print the results.

Step 5:Stop the program.

**PROGRAM**

import java.io.\*;

import java.net.\*;

class PingServer

{

public static void main(String args[])

{

try

{

String str;

System.out.print(" Enter the IP Address to be Ping : ");

BufferedReader buf1=new BufferedReader(newInputStreamReader(System.in));

String ip=buf1.readLine();

Runtime H=Runtime.getRuntime();

Process p=H.exec("ping " + ip);

InputStream in=p.getInputStream();

BufferedReader buf2=new BufferedReader(newInputStreamReader(in));

while((str=buf2.readLine())!=null)

{

System.out.println(" " + str);

}

}

catch(Exception e)

{

System.out.println(e.getMessage());

}

}

}

**OUTPUT**

Enter the IP address to the ping:192.168.0.1

Pinging 192.168.0.1: with bytes of data =32

Reply from 192.168.0.1:bytes=32 time<1ms TTL =128

Reply from 192.168.0.1:bytes=32 time<1ms TTL =128

Reply from 192.168.0.1:bytes=32 time<1ms TTL =128

Reply from 192.168.0.1:bytes=32 time<1ms TTL =128

Ping statistics for 192.168.0.1

Packets: Sent=4, Received=4, Lost=0(0% loss),

Approximate round trip time in milli-seconds:

Minimum=1ms, Maximum=4ms, Average=2ms

**RESULT**

Thus the program to simulate PING Command is implemented successfully.

**EX-NO 4(b) Implementation ofTRACEROUTE Command**

**Date**

**AIM**

To write a java program for simulating Traceroute Command

**ALGORITHM**

Step 1: start the program.

Step 2: Include necessary package in java.

Step 3: Create a process object to implement the tracert command.

Step 4: Execute the system command and print the results.

Step 5:Stop the program.

**PROGRAM**

import java.io.\*;

import java.net.\*;

class TracertServer

{

public static void main(String args[])

{

try

{

String str;

System.out.print(" Enter the Domain to trace : ");

BufferedReader buf1=new BufferedReader(newInputStreamReader(System.in));

String domain=buf1.readLine();

Runtime H=Runtime.getRuntime();

Process p=H.exec("tracert " +domain);

InputStream in=p.getInputStream();

BufferedReader buf2=new BufferedReader(newInputStreamReader(in));

while((str=buf2.readLine())!=null)

{

System.out.println(" " + str);

}

}

catch(Exception e)

{

System.out.println(e.getMessage());

}

}

}

**OUTPUT**

Enter the Domain to trace : www.google.com

Tracing route to www.google.com [172.217.163.132]

over a maximum of 30 hops:

1 2 ms 3 ms 1 ms 192.168.0.1

2 8 ms 9 ms 9 ms 121.200.54.1

3 19 ms 19 ms 20 ms 209.85.174.10

4 19 ms 19 ms 19 ms 74.125.242.145

5 20 ms 19 ms 19 ms 216.239.42.245

6 20 ms 19 ms 19 ms maa05s04-in-f4.1e100.net [172.217.163.132]

Trace complete.

**RESULT**

Thus the program to simulate TRACEROUTE Command is implemented successfully.

**EX.NO 5 Implementation of Web Page Upload and Download**

**Date:**

**AIM**

To write a java program for web page upload and download using socket for HTTP.

**ALGORITHM**

1.Start the program.

2. Read the image from the system.

3. Send the image to the server.

4. At server side receive the image sent by the client.

5. Print the image size.

6. Stop the program.

**PROGRAM**

**Client**

import javax.swing.\*;

import java.net.\*;

import java.awt.image.\*;

import javax.imageio.\*;

import java.io.\*;

import java.awt.image.BufferedImage;

import java.io.ByteArrayOutputStream;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

public class Client

{

public static void main(String args[]) throws Exception

{

Socket soc;

BufferedImage img = null;

soc=new Socket("localhost",4000);

System.out.println("Client is running. ");

try

{

System.out.println("Reading image from disk. ");

img = ImageIO.read(new File("network.jpg"));

ByteArrayOutputStream baos = new ByteArrayOutputStream();

ImageIO.write(img, "jpg", baos);

baos.flush();

byte[] bytes = baos.toByteArray(); baos.close();

System.out.println("Sending image to server. ");

OutputStream out = soc.getOutputStream();

DataOutputStream dos = new DataOutputStream(out);

dos.writeInt(bytes.length);

dos.write(bytes, 0, bytes.length);

System.out.println("Image sent to server. ");

dos.close();

out.close();

}

catch (Exception e)

{

System.out.println("Exception: " + e.getMessage());

soc.close();

}

soc.close();

}

}

**Server**

import java.net.\*;

import java.io.\*;

import java.awt.image.\*;

import javax.imageio.\*;

import javax.swing.\*;

class Server

{

public static void main(String args[]) throws Exception

{

ServerSocket server=null;

Socket socket;

server=new ServerSocket(4000);

System.out.println("Server Waiting for image");

socket=server.accept();

System.out.println("Client connected.");

InputStream in = socket.getInputStream();

DataInputStream dis = new DataInputStream(in);

int len = dis.readInt();

System.out.println("Image Size: " + len/1024 + "KB"); byte[] data = new byte[len];

dis.readFully(data);

dis.close();

in.close();

InputStream ian = new ByteArrayInputStream(data);

BufferedImage bImage = ImageIO.read(ian);

JFrame f = new JFrame("Server");

ImageIcon icon = new ImageIcon(bImage);

JLabel l = new JLabel();

l.setIcon(icon);

f.add(l);

f.pack();

f.setVisible(true);

}

}

**OUTPUT**

**OUTPUT FOR CLIENT**

Client is running.

Reading image from disk.

Sending image to server.

Image sent to server.

**OUTPUT FOR SERVER**

Sever Waiting for image

Client connected

Image Size: 30KB

**RESULT**

Thus the program to simulate TRACEROUTE Command is implemented successfully.

**EX-NO 6Implementation of RPC(Remote Procedure Call)**

**Date**

**AIM**

To write a java program to implement RPC (remote procedure call)

**ALGORITHM**

1. Start the program.

2. Include the necessary packages in java.

3. Create an interface by extendingRemote and declare methods.

4. Create a RPC class that implements the interface and override the methods.

5. Create a server class and bind it with the RPC class.

6. Create a client class and invoke the methods.

7. Stop the Program

**PROGRAM**

**Client**

import java.rmi.\*;

import java.io.\*;

import java.rmi.server.\*;

public class ClientRPC

{

public static void main(String arg[])

{

try

{

String serverurl="rmi://localhost/ServerRPC";

UCET ob=(UCET) Naming.lookup(serverurl);

int r=ob.function(10,5);

System.out.println("the answer of(a+b)^2 is:"+r);

int t =ob.power(10,5);

System.out.println("the answer of(a)^(b) is:"+t);

double d=ob.log(10);

System.out.println("the log value of the given number "+10+" is :"+d);

}

catch(Exception e)

{

System.out.println("error.."+e.getMessage());

}

}

}

**Server**

import java.rmi.\*;

import java.rmi.server.\*;

public class ServerRPC

{

public static void main(String arg[])

{

try

{

RPC ob=new RPC();

Naming.rebind("ServerRPC",ob);

}

catch(Exception e)

{

}

}}

**RPC**

import java.rmi.\*;

import java.lang.Math.\*;

import java.rmi.server.\*;

public class RPC extends UnicastRemoteObject implements UCET

{

public RPC()throws Exception

{

}

public int function(int a,int b)throws RemoteException

{

int m;

m=(a\*a)+(b\*b)+(2\*a\*b);

return m;

}

public int power(int a,int b)throws RemoteException

{

int m=(int)Math.pow(a,b);

return m;

}

public double log(int a)throws RemoteException

{

return(Math.log(a));

}

}

**UCET**

import java.rmi.\*;

public interface UCET extends Remote

{

public int function(int a,int b)throws RemoteException;

public int power(int a,int b)throws RemoteException;

public double log(int a)throws RemoteException;

}

**OUTPUT**

**OUTPUT FOR SERVER**

javac UCET.java

start rmiregistry

javac rpc.java

rmic rpc

javac serverrpc.java

java serverrpc

**OUTPUT FOR CLIENT**

javac clientrpc.java

java clientrpc

the ans of (a+b)^2 is:225

the ans of (a)^(b) is :100000

the log value of the given number 10 is 2.30258

**RESULT**

Thus the implementation of RPC (Remote Procedure Call)is done successfully

**EX-NO 7Implementation ofSubnetting**

**Date**

**AIM**

Write a program to implement Subnetting and find the subnet masks.

**ALGORITHM**

1.Start the program.

2. Read the IP address from the user.

3. Convert the string to number and then to binary.

4. Print the binary for of the address.

5. Calculate the mask.

6. Calculate first address and last address

7. Get first address by ANDing last n bits with 0

8. Get last address by ORing last n bits with 1

9. Print result.

10. Stop the program.

**PROGRAM**

import java.util.Scanner;

class Subnet

{

public static void main(String args[])

{

Scanner sc = new Scanner(System.in);

System.out.print("Enter the ip address");

String ip = sc.nextLine();

String split\_ip[]=ip.split("\\.");

String split\_bip[] = new String[4];

String bip = "";

for(int i=0;i<4;i++){

split\_bip[i] = appendZeros(Integer.toBinaryString(Integer.parseInt(split\_ip[i])));

bip += split\_bip[i];

}

System.out.println("IP in binary is "+bip);

System.out.print("Enter the number of addresses: ");

int n = sc.nextInt();

int bits = (int)Math.ceil(Math.log(n)/Math.log(2));

System.out.println("Number of bits required for address = "+bits);

int mask = 32-bits;

System.out.println("The subnet mask is = " +mask);

int fbip[] = new int[32];

for(int i=0; i<32;i++)

fbip[i] = (int)bip.charAt(i)-48;

for(int i=31;i>31-bits;i--)

fbip[i] &= 0;

String fip[] = {"","","",""};

for(int i=0;i<32;i++)

fip[i/8] = new String(fip[i/8]+fbip[i]);

System.out.print("First address is = ");

for(int i=0;i<4;i++)

{

System.out.print(Integer.parseInt(fip[i],2));

if(i!=3)

System.out.print(".");

}

System.out.println();

int lbip[] = new int[32];

for(int i=0; i<32;i++) lbip[i] = (int)bip.charAt(i)-48;

for(int i=31;i>31-bits;i--)

lbip[i] |= 1;

String lip[] ={"","","",""}; for(int i=0;i<32;i++)

lip[i/8] = new String(lip[i/8]+lbip[i]); System.out.print("Last address is =");

for(int i=0;i<4;i++){ System.out.print(Integer.parseInt(lip[i],2)); if(i!=3) System.out.print(".");

}

System.out.println();

}

static String appendZeros(String s){

String temp = new String("00000000"); return temp.substring(s.length())+ s;

}

}

**OUTPUT**

Enter the ip address: 100.110.150.10

IP in binary is 01100100011011101001011000001010

Enter the number of addresses: 7

Number of bits required for address = 3

The subnet mask is = 29

First address is = 100.110.150.8

Last address is = 100.110.150.15

**RESULT**

Thus the implementation of Subnettingis done successfully.

**EX-NO 8 Applications Using TCP Socket**

**Date**

1. **Echo client and Echo server**

**AIM**

To write a java program to implementEcho client and Echo server.

**ALGORITHM**

1.Start the program.

2. Create a server socket and bind it to port.

3. Listen for new connection and when a connection arrives, accept it.

4. At client side read input from the user.

5. Send it to the server and receive echoed data from the server.

6. Print received data on console.

7. At Server side receive and resend the data sent by the client.

8. Stop the program.

**PROGRAM**

**EchoServer**

import java.io.\*;

import java.net.\*;

class EServer

{

public static void main(String args[])

{

ServerSocket s=null;

String line;

DataInputStream is;

PrintStream ps;

Socket c=null;

try

{

s=new ServerSocket(9000);

}

catch(IOException e)

{

System.out.println(e);

}

try

{

c=s.accept();

is=new DataInputStream(c.getInputStream());

ps=new PrintStream(c.getOutputStream());

while(true)

{

line=is.readLine();

ps.println(line);

}

}

catch(IOException e)

{

System.out.println(e);

}

}

}

**EchoClient**

import java.io.\*;

import java.net.\*;

class EClient

{

public static void main(String arg[])

{

Socket c=null;

String line;

DataInputStream is,is1;

PrintStream os;

try

{

InetAddress ia = InetAddress.getLocalHost();

c=new Socket(ia,9000);

}

catch(IOException e)

{

System.out.println(e);

}

try

{

os=new PrintStream(c.getOutputStream());

is=new DataInputStream(System.in);

is1=new DataInputStream(c.getInputStream());

while(true)

{

System.out.print("Client:");

line=is.readLine();

os.println(line);

System.out.println("Server:" + is1.readLine());

}}

catch(IOException e)

{

System.out.println("Socket Closed!");

}

}

}

**OUTPUT**

Client:Hai Server

Server:Hai Server

Client:Hello

Server:Hello

Client:end

Server:end

1. **Chat**

**AIM**

To write a java program to implementChat client and Chat server.

**ALGORITHM**

1.Start the program.

2. Create a server socket and bind it to port.

3. Listen for new connection and when a connection arrives, accept it.

4. At client side read input from the user.

5. Send it to the server and receive data from the server.

6. Print received data on console.

7. At Server side receive input from the client and print it on the console.

8. Read reply from the user and send it to the client.

9. Stop the program.

**PROGRAM**

**ChatServer**

import java.io.\*;

import java.net.\*;

class Server

{

public static DatagramSocket ds;

public static byte buffer[]=new byte[1024];

public static int clientport=789,serverport=790;

public static void main(String args[])throws Exception

{

ds=new DatagramSocket(clientport);

System.out.println("press ctrl+c to quit the program");

BufferedReader dis=new BufferedReader(new InputStreamReader(System.in));

InetAddress ia=InetAddress.getLocalHost();

while(true)

{

DatagramPacket p=new DatagramPacket(buffer,buffer.length);

ds.receive(p);

String psx=new String(p.getData(),0,p.getLength());

System.out.println("Client:" + psx);

System.out.println("Server:");

String str=dis.readLine();

if(str.equals("end"))

break;

buffer=str.getBytes();

ds.send(new

DatagramPacket(buffer,str.length(),ia,serverport));

}

}

}

**ChatClient**

import java .io.\*;

import java.net.\*;

class Client

{

public static DatagramSocket ds;

public static int clientport=789,serverport=790;

public static void main(String args[])throws Exception

{

byte buffer[]=new byte[1024];

ds=new DatagramSocket(serverport);

BufferedReader dis=new BufferedReader(new InputStreamReader(System.in));

System.out.println("server waiting");

InetAddress ia=InetAddress.getLocalHost();

while(true)

{

System.out.println("Client:");

String str=dis.readLine();

if(str.equals("end"))

break;

buffer=str.getBytes();

ds.send(new DatagramPacket(buffer,str.length(),ia,clientport));

DatagramPacket p=new DatagramPacket(buffer,buffer.length);

ds.receive(p);

String psx=new String(p.getData(),0,p.getLength());

System.out.println("Server:" + psx);

}

}

}

**OUTPUT**

**OUTPUT FOR SERVER**

press ctrl+c to quit the program

Client:Hai Server

Server:

Hello Client

Client:How are You

Server:

I am Fine

**OUTPUT FOR CLIENT**

server waiting

Client:

Hai Server

Server:Hello Clie

Client:

How are You

Server:I am Fine

1. **File Transfer**

**AIM**

To write a java program to implementFile Transfer.

**ALGORITHM**

1.Start the program.

2. Create a server socket and bind it to port.

3. Listen for new connection and when a connection arrives, accept it.

4. At client side read input file name and new file name from the user.

5. Send it to the server and receive data from the server.

6. Write the content in console and in the new file.

7. At Server side receive input file name from the client.

8. Read content from the file.

9. Write the content in console and in the socket which is connected to the client.

10. Stop the program.

**PROGRAM**

**File Client**

import java.io.\*;

import java.net.\*;

import java.util.\*;

class Clientfile

{ public static void main(String args[])

{

try

{

BufferedReader in=new BufferedReader(newInputStreamReader(System.in)); Socket clsct=new Socket("127.0.0.1",131);

DataInputStream din=new DataInputStream(clsct.getInputStream());

DataOutputStream dout=new DataOutputStream(clsct.getOutputStream());

System.out.println("Enter the file name:");

String str=in.readLine();

dout.writeBytes(str+'\n');

System.out.println("Enter the new file name:");

String str2=in.readLine();

String str1,ss;

FileWriter f=newFileWriter(str2);

char buffer[];

while(true)

{

str1=din.readLine();

if(str1.equals("-1")) break;

System.out.println(str1);

buffer=new char[str1.length()];

str1.getChars(0,str1.length(),buffer,0);

f.write(buffer);

}

f.close();

clsct.close();

}

catch (Exception e)

{

System.out.println(e);

}}}

**File server**

import java.io.\*;

import java.net.\*;

import java.util.\*;

class Serverfile

{ public static void main(String args[])

{

try

{

ServerSocket obj=new ServerSocket(131);

while(true)

{

Socket obj1=obj.accept();

DataInputStream din=new DataInputStream(obj1.getInputStream());

DataOutputStream dout=new DataOutputStream(obj1.getOutputStream());

String str=din.readLine();

FileReader f=new FileReader(str);

BufferedReader b=new BufferedReader(f);

String s;

while((s=b.readLine())!=null) {

System.out.println(s);

dout.writeBytes(s+'\n');

}

f.close();

dout.writeBytes("-1\n");

} }

catch(Exception e)

{ System.out.println(e);}

}

}

**OUTPUT**

File content src.txt

Computer networks

OSI Model

Sliding Window Protocols

Routing

**OUTPUT FOR CLIENT**

Enter the file name:

src.txt

Enter the new file name:

dest.txt

Computer networks

OSI Model

Sliding Window Protocols

Routing

**OUTPUT FOR SERVER**

Computer networks

OSI Model

Sliding Window Protocols

Routing

**Result**:

Thus the applications using TCP Sockets like Echo client and Echo server, Chat andFile Transfer are done successfully.

**EX-NO 9 Applications using TCP and UDP Sockets like DNS, SNMP and File**

**Date:**

1. **DNS**

**AIM**

To write a java program for DNS application

**ALGORITHM**

**Client**

1. Start the program

2. Using Socket, connection is established between client and server.

3. Get the website name to be converted into IP address.

4. Send this name to server.

5**.** Read IP from server and print in the console.

**Server**

1. Start the program

2. Accept the socket which is created by the client.

3. Server maintains the table in which domain name and corresponding IP addresses are stored.

4. Read the domain name which is sent by the client.

5. Map the domain name with its IP address and return the IP address to client.

**PROGRAM**

**UDP DNS Server**

import java.io.\*;

import java.net.\*;

public class Udpdnsserver

{

private static int indexOf(String[] array, String str)

{

str = str.trim();

for (int i=0; i < array.length; i++)

{

if (array[i].equals(str)) return i;

}

return -1;

}

public static void main(String arg[])throws IOException

{

String[] hosts = {"yahoo.com", "gmail.com","cricinfo.com", "facebook.com"};

String[] ip = {"68.180.206.184", "209.85.148.19","80.168.92.140", "69.63.189.16"};

System.out.println("Press Ctrl + C to Quit");

while (true)

{

DatagramSocket serversocket=new DatagramSocket(1362);

byte[] senddata = new byte[1021];

byte[] receivedata = new byte[1021];

DatagramPacket recvpack = new DatagramPacket(receivedata, receivedata.length);

serversocket.receive(recvpack);

String sen = new String(recvpack.getData());

InetAddress ipaddress = recvpack.getAddress();

int port = recvpack.getPort();

String capsent;

System.out.println("Request for host " + sen);

if(indexOf (hosts, sen) != -1)

capsent = ip[indexOf (hosts, sen)];

else capsent = "Host Not Found";

senddata = capsent.getBytes();

DatagramPacket pack = new DatagramPacket (senddata, senddata.length,ipaddress,port);

serversocket.send(pack);

serversocket.close();

}

}

}

**UDP DNS Client**

import java.io.\*;

import java.net.\*;

public class Udpdnsclient

{

public static void main(String args[])throws IOException

{

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

DatagramSocket clientsocket = new DatagramSocket();

InetAddress ipaddress;

if (args.length == 0)

ipaddress =InetAddress.getLocalHost();

else

ipaddress = InetAddress.getByName(args[0]);

byte[] senddata = new byte[1024];

byte[] receivedata = new byte[1024];

int portaddr = 1362;

System.out.print("Enter the hostname : ");

String sentence = br.readLine();

senddata = sentence.getBytes();

DatagramPacket pack = new DatagramPacket(senddata,senddata.length, ipaddress,portaddr);

clientsocket.send(pack);

DatagramPacket recvpack =new DatagramPacket(receivedata,receivedata.length);

clientsocket.receive(recvpack);

String modified = new String(recvpack.getData());

System.out.println("IP Address: " + modified);

clientsocket.close();

}

}

**OUTPUT**

**OUTPUT FOR SERVER**

Press Ctrl + C to Quit

Request for host yahoo.com

Request for host cricinfo.com

Request for host youtube.com

**OUTPUT FOR CLIENT**

Enter the hostname : yahoo.com

IP Address: 68.180.206.184

java udpdnsclient

Enter the hostname : cricinfo.com

IP Address: 80.168.92.140

java udpdnsclient

Enter the hostname : youtube.com

IP Address: Host Not Found

**B.SNMP**

**Aim**

To write a java program for SNMP application program

**ALGORITHM**

**Client**

1. Start the program

2. Using Socket, connection is established between client and server.

3. Get the server parameter.

4. Send this name to server.

5**.** Read value from server and print in the console.

**Server**

1. Start the program

2. Accept the socket which is created by the client.

3. Server maintains the table in which parameter name and value are stored.

4. Read the parameter name which is sent by the client.

5. Map the name with its value and return the IP address to client.

**PROGRAM**

**Client**

import java.net.\*;

import java.io.\*;

public class snmpclient

{

public static void main(String args[])throws IOException

{

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

DatagramSocket clientsocket = new DatagramSocket();

InetAddress ipaddress;

if (args.length == 0)

ipaddress =InetAddress.getLocalHost();

else

ipaddress = InetAddress.getByName(args[0]);

byte[] senddata = new byte[1024];

byte[] receivedata = new byte[1024];

int portaddr = 1362;

System.out.print("Enter the parameters : ");

String sentence = br.readLine();

senddata = sentence.getBytes();

DatagramPacket pack = new DatagramPacket(senddata,senddata.length, ipaddress,portaddr);

clientsocket.send(pack);

DatagramPacket recvpack =new DatagramPacket(receivedata,receivedata.length);

clientsocket.receive(recvpack);

String modified = new String(recvpack.getData());

System.out.println(modified);

clientsocket.close();

}

}

**Server**

import java.net.\*;

import java.io.\*;

public class snmpserver

{

private static int indexOf(String[] array, String str)

{

str = str.trim();

for (int i=0; i < array.length; i++)

{

if (array[i].equals(str)) return i;

}

return -1;

}

public static void main(String arg[])throws IOException

{

String[] hosts = {"temperature", "number of packets"};

String[] ip = {"40c", "100"};

System.out.println("Press Ctrl + C to Quit");

while (true)

{

DatagramSocket serversocket=new DatagramSocket(1362);

byte[] senddata = new byte[1021];

byte[] receivedata = new byte[1021];

DatagramPacket recvpack = new DatagramPacket(receivedata, receivedata.length);

serversocket.receive(recvpack);

String sen = new String(recvpack.getData());

InetAddress ipaddress = recvpack.getAddress();

int port = recvpack.getPort();

String capsent;

System.out.println("Request for host " + sen);

if(indexOf (hosts, sen) != -1)

capsent = ip[indexOf (hosts, sen)];

else capsent = "Host Not Found";

senddata = capsent.getBytes();

DatagramPacket pack = new DatagramPacket (senddata, senddata.length,ipaddress,port);

serversocket.send(pack);

serversocket.close();

}

}

}

**OUTPUT**

**OUTPUT FOR SERVER**

Press Ctrl + C to Quit

Request for temperature

Request for number of packets

**OUTPUT FOR CLIENT**

Enter the Parameters : temperature

40c

Enter the Parameters : number of packets

100

**C. File Transfer**

**AIM**

To write a java program for applaction using TCP and UDP Sockets Links

**ALGORITHM**

1.Start the program.

2. Create a server socket and bind it to port.

3. Listen for new connection and when a connection arrives, accept it.

4. At client side read input file name and new file name from the user.

5. Send it to the server and receive data from the server.

6. Write the content in console and in the new file.

7. At Server side receive input file name from the client.

8. Read content from the file.

9. Write the content in console and in the socket which is connected to the client.

10. Stop the program.

**PROGRAM**

**Client**

import java.io.\*;

import java.net.\*;

import java.util.\*;

class Clientfile

{

public static void main(String args[])

{

try{

BufferedReader in=new BufferedReader(new InputStreamReader(System.in));

Socket clsct=new Socket("127.0.0.6",253);

DataInputStream din=new DataInputStream(clsct.getInputStream());

DataOutputStream dout=new DataOutputStream(clsct.getOutputStream());

System.out.println("Enter the file name:");

String str=in.readLine();

dout.writeBytes(str+'\n');

System.out.println("Enter the new file name:");

String str2=in.readLine();

String str1,ss;

FileWriter f=new

FileWriter(str2);

char buffer[];

while(true){

str1=din.readLine();

if(str1.equals("-1")) break;

System.out.println(str1);

buffer=new char[str1.length()];

str1.getChars(0,str1.length(),buffer,0);

f.write(buffer);

}

f.close();

clsct.close();

}

catch (Exception e)

{

System.out.println(e);

}}}

**Server**

import java.io.\*;

import java.net.\*;

import java.util.\*;

class Serverfile

{ public static void main(String args[])

{

try

{

ServerSocket obj=new ServerSocket(253);

while(true)

{

Socket obj1=obj.accept();

DataInputStream din=new DataInputStream(obj1.getInputStream());

DataOutputStream dout=new DataOutputStream(obj1.getOutputStream());

String str=din.readLine();

FileReader f=new FileReader(str);

BufferedReader b=new BufferedReader(f);

String s;

while((s=b.readLine())!=null) {

System.out.println(s);

dout.writeBytes(s+'\n');

}

f.close();

dout.writeBytes("-1\n");

} }

catch(Exception e)

{ System.out.println(e);}

}

}

**OUTPUT**

File content src.txt

Computer networks

OSI Model

Sliding Window Protocols

Routing

**OUTPUT FOR CLIENT**

Enter the file name:

src.txt

Enter the new file name:

dest.txt

Computer networks

OSI Model

Sliding Window Protocols

Routing

**OUTPUT FOR SERVER**

Computer networks

OSI Model

Sliding Window Protocols

Routing

**RESULT**

Thus the implementation of applications using UDP Sockets like DNS, SNMP and FileTransfer are done successfully.

**EX-NO 10 Study of Network Simulator and Simulation of Congestion Control Algorithms using NS**

**Aim:**

To Study of Network simulator (NS).and Simulation of Congestion Control Algorithms using NS

**NET WORK SIMULATOR (NS2)**

**NS2**

NS2 stands for Network Simulator Version 2. It is an open-source event-driven simulator designed specifically for research in computer communication networks.

**Features of NS2**

1. It is a discrete event simulator for networking research.

2. It provides substantial support to simulate bunch of protocols like TCP, FTP, UDP, https and DSR.

3. It simulates wired and wireless network.

4. It is primarily Unix based.

5. Uses TCL as its scripting language.

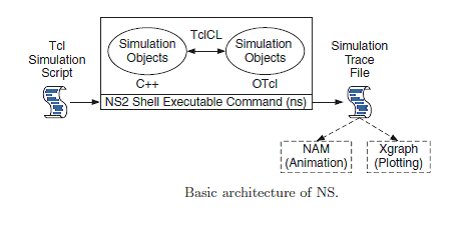
6. Otcl: Object oriented support

7. Tclcl: C++ and otcl linkage

8. Discrete event scheduler

**Basic Architecture**

NS2 consists of two key languages: C++ and Object-oriented Tool Command Language (OTcl). While the C++ defines the internal mechanism (i.e., a backend) of the simulation objects, the OTcl sets up simulation by assembling and configuring the objects as well as scheduling discrete events. The C++ and the OTcl are linked together using TclCL



## TCL and C++

NS2 stands for Network Simulator Version 2. It is an open-source event-driven simulator designed specifically for research in computer communication networks.

NS2 uses OTcl to create and configure a network, and uses C++ to run simulation. All C++ codes need to be compiled and linked to create an executable file.

**Use OTcl**

1. For configuration, setup, or one time simulation, or
2. To run simulation with existing NS2 modules.

This option is preferable for most beginners, since it does not involve complicated internal mechanism of NS2. Unfortunately, existing NS2 modules are fairly limited. This option is perhaps not sufficient for most researchers.

**Use C++**

1. When you are dealing with a packet, or - when you need to modify existing NS2 modules.

This option perhaps discourages most of the beginners from using NS2. This book particularly aims at helping the readers understand the structure of NS2 and feel more comfortable in modifying NS2 modules.

### Tools for generating TCL Script for NS2

NS2 a very common and widely used tool to simulate small and large area networks. Tcl scripts are widely used in NS-2 simulation tool. Tcl scripts are used to set up a wired or wireless communication network, and then run these scripts via the NS-2 for getting the simulation results.Several tools are available to design networks and generate TCL scripts some of them are discussed below

**I. NS2 scenario Generator (NSG)**

Its a java based tool that can run on any platform and can generate TCL scripts for wired and Wireless scenarios for NS2.Main features of NSG are:

1. Creating Wired and wireless nodes by drag and drop.
2. Creating Simplex and Duplex links for wired network
3. Creating Grid, Random and Chain topologies.
4. Creating TCP and UDP agents. Also supports TCP
5. Tahoe, TCP Reno, TCP New-Reno and TCP Vegas.
6. Supports Ad Hoc routing protocols such as DSDV,
7. AODV, DSR and TORA.
8. Supports FTP and CBR applications.
9. Supports node mobility.
10. Setting the packet size, start time of simulation, end
11. Time of simulation, transmission range and interference
12. Range in case of wireless networks, etc.
13. Setting other network parameters such as bandwidth, etc for wireless scenarios

**II. Visual Network Simulator (VNS)**

This tool is centered on capabilities of NSG. It also provides support to Differentiated Services (DiffServ) scenarios and simple and intuitive set of icons to represent the components of a network. Some features of VNS are given below:

1. Adding and configuration of links, agents and traffic sources.
2. Modeling network scenarios with support to multicast.
3. Selection of a dynamic routing protocol.
4. Definition of the simulation output as an animation and/or graphics.
5. Edition of the Tcl script generated.
6. Saving the defined simulation scenario

**III. NS 2 Workbench**

Ns Bench makes NS-2 simulation development and analysis faster and easier for students and researchers without losing the flexibility or expressiveness gained by writing a script. Some features are:

1. Nodes, simplex/duplex links and LANs
2. Agents: TCP,UDP, TCPSink, TCP/Fack,TCP/FullTcp, TCP/Newreno, TCP/Reno,TCP/Sack1, TCPSink, TCPSink/Sack1,TCPSink/DelAck,
3. TCPSink/Sack1/DelAck,TCP/Vegas, Null Agent.
4. Applications/Traffic: FTP, Telent, https/Server,https/Client, https/Cache, webtraf, Traffic/CBR,Traffic/Pareto, Traffic/Exponential.
5. Services: Multicast, Packet Scheduling, RED, Diff-Serv.
6. Creating "Groups" concept to compensate for "loops"
7. Scenario generator.
8. Link Monitors.
9. Loss Models.
10. Routing Protocols

**IV. Network Simulation by Mouse (NSBM)**

NSBM, developed in java, is a graphical tool that is used to generate TCL script using a mouse. Nodes and links can be created with a single mouse click. You can draw a network topology with multiple nodes with only a few mouse clicks. Afterwards you click on a button and there is the TCL code, almost ready for use with the ns.

NSBM used in order to process the XML configuration data. It must provide many functions, which are specified only in the configuration data at run time. Because the classes are implementation-specific, classes generated by the binding compiler in one JAXB implementation will probably not work with another JAXB implementation. So if you change to another JAXB implementation, you should rebind the schema with the binding compiler provided by that implementation

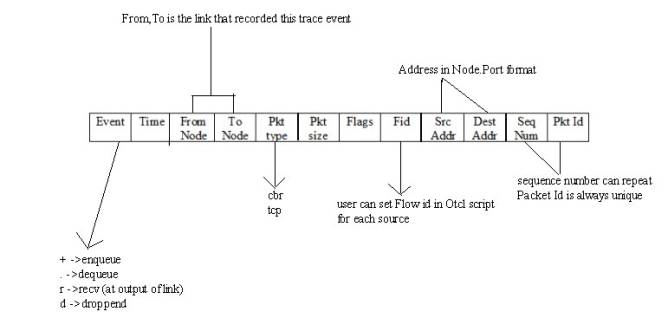
### Trace Files Generated in NS2

NS2 currently supports a number of different types of trace files. In addition to its own format, NS2 also has the Nam trace format, which contains the necessary information from the simulation to drive the Nam visualizer. Both of these trace formats are very specific when it comes to giving details about the events that occur during an NS2 simulation.

Traces and monitors represent the only support for data collection in ns-2. Traces record events related to the generation, enqueueing, forwarding, and dropping of packets. Each event corresponds to a line of ASCII characters, which contains information on the event type and the information stored into the packet

NS-2 provides three kinds of formats for wired networks: Tracing, Monitoring and NAM trace file.

I. Tracing: Trace file format is given below:

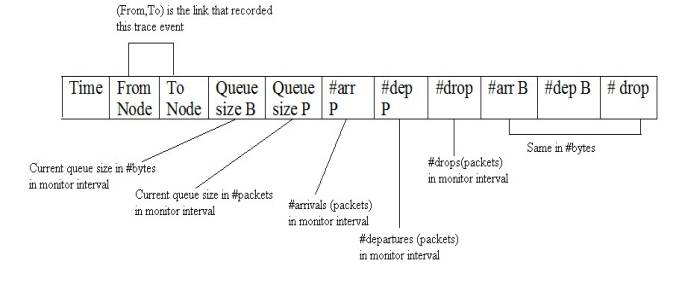


1. Operation performed in the simulation
2. Simulation time of event occurrence
3. Node 1 of what is being traced
4. Node 2 of what is being traced
5. Packet type
6. Packet size
7. Flags
8. IP flow identifier
9. Packet source node address
10. Packet destination node address
11. Sequence number
12. Unique packet identifier

II. Monitoring

Queue monitoring refers to the capability of tracking the dynamics of packets at a queue (or other object). A queue monitor tracks packet arrival/departure/drop statistics, and may optionally compute averages of these values. Monitoring was useful tools to find detail information about queue.

Flow monitor trace format is given below:

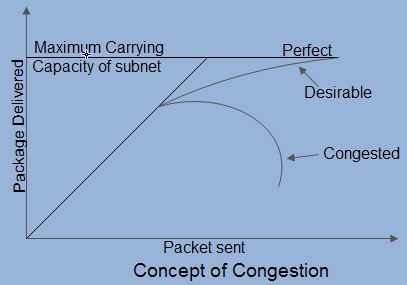


III. NAM trace files which are used by NAM for visualization of ns simulations. The NAM trace file should contain topology information like nodes, links, queues, node connectivity etc as well as packet trace information. A NAM trace file has a basic format to it. Each line is a NAM event. The first character on the line defines the type of event and is followed by several flags to set options on that event. There are 2 sections in that file, static initial configuration events and animation events. All events with -t \* in them are configuration events and should be at the beginning of the file.

**CONGESTION CONTROL ALGORITHMS**

Congestion is an important issue that can arise in packet switched network. Congestion is a situation in Communication Networks in which too many packets are present in a part of the subnet, performance degrades. Congestion in a network may occur when the load on the network *(i.e.*the number of packets sent to the network) is greater than the capacity of the network *(i.e.*the number of packets a network can handle.). Network congestion occurs in case of traffic overloading.

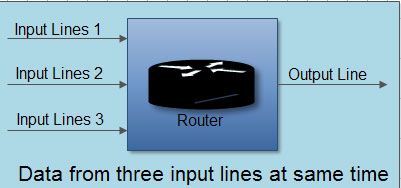
In other words when too much traffic is offered, congestion sets in and performance degrades sharply

[](http://ecomputernotes.com/images/Concept-of-Congestion.jpg)

## Causing of Congestion:

The various causes of congestion in a subnet are:

• The input traffic rate exceeds the capacity of the output lines. If suddenly, a stream of packet start arriving on three or four input lines and all need the same output line. In this case, a queue will be built up. If there is insufficient memory to hold all the packets, the packet will be lost. Increasing the memory to unlimited size does not solve the problem. This is because, by the time packets reach front of the queue, they have already timed out (as they waited the queue). When timer goes off source transmits duplicate packet that are also added to the queue. Thus same packets are added again and again, increasing the load all the way to the destination.

[](http://ecomputernotes.com/images/Data-from-three-input-lines-at-same-time.jpg)

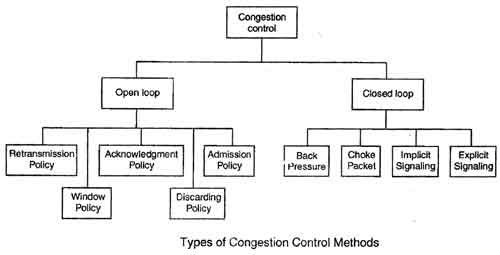
The routers are too slow to perform bookkeeping tasks (queuing buffers, updating tables, etc.)

The routers’ buffer is too limited.

Congestion in a subnet can occur if the processors are slow. Slow speed CPU at routers will perform the routine tasks such as queuing buffers, updating table etc slowly. As a result of this, queues are built up even though there is excess line capacity.  
Congestion is also caused by slow links. This problem will be solved when high speed links are used. But it is not always the case. Sometimes increase in link bandwidth can further deteriorate the congestion problem as higher speed links may make the network more unbalanced.Congestion can make itself worse. If a route!" does not have free buffers, it start ignoring/discarding the newly arriving packets. When these packets are discarded, the sender may retransmit them after the timer goes off. Such packets are transmitted by the sender again and again until the source gets the acknowledgement of these packets. Therefore multiple transmissions of packets will force the congestion to take place at the sending end.

## How to correct the Congestion Problem:

Congestion Control refers to techniques and mechanisms that can either prevent congestion, before it happens, or remove congestion, after it has happened. Congestion control mechanisms are divided into two categories, one category prevents the congestion from happening and the other category removes congestion after it has taken place.



These two categories are:

1. Open loop

2. Closed loop

## Open Loop Congestion Control

• In this method, policies are used to prevent the congestion before it happens.

• Congestion control is handled either by the source or by the destination.

• The various methods used for open loop congestion control are:

### Retransmission Policy

• The sender retransmits a packet, if it feels that the packet it has sent is lost or corrupted.

• However retransmission in general may increase the congestion in the network. But we need to implement good retransmission policy to prevent congestion.

• The retransmission policy and the retransmission timers need to be designed to optimize efficiency and at the same time prevent the congestion.

### Window Policy

• To implement window policy, selective reject window method is used for congestion control.

• Selective Reject method is preferred over Go-back-n window as in Go-back-n method, when timer for a packet times out, several packets are resent, although some may have arrived safely at the receiver. Thus, this duplication may make congestion worse.

• Selective reject method sends only the specific lost or damaged packets.

### Acknowledgement Policy

• The acknowledgement policy imposed by the receiver may also affect congestion.

• If the receiver does not acknowledge every packet it receives it may slow down the sender and help prevent congestion.

• Acknowledgments also add to the traffic load on the network. Thus, by sending fewer acknowledgements we can reduce load on the network.

• To implement it, several approaches can be used:

1. A receiver may send an acknowledgement only if it has a packet to be sent.

2. A receiver may send an acknowledgement when a timer expires.

3. A receiver may also decide to acknowledge only *N*packets at a time.

### Discarding Policy

• A router may discard less sensitive packets when congestion is likely to happen.

• Such a discarding policy may prevent congestion and at the same time may not harm the integrity of the transmission.

### Admission Policy

• An admission policy, which is a quality-of-service mechanism, can also prevent congestion in virtual circuit networks.

• Switches in a flow first check the resource requirement of a flow before admitting it to the network.

• A router can deny establishing a virtual circuit connection if there is congestion in the "network or if there is a possibility of future congestion.

## Closed Loop Congestion Control

• Closed loop congestion control mechanisms try to remove the congestion after it happens.

• The various methods used for closed loop congestion control are:

### Backpressure

• Backpressure is a node-to-node congestion control that starts with a node and propagates, in the opposite direction of data flow.

• The backpressure technique can be applied only to virtual circuit networks. In such virtual circuit each node knows the upstream node from which a data flow is coming.

• In this method of congestion control, the congested node stops receiving data from the immediate upstream node or nodes.

This may cause the upstream node on nodes to become congested, and they, in turn, reject data from their upstream node or nodes.

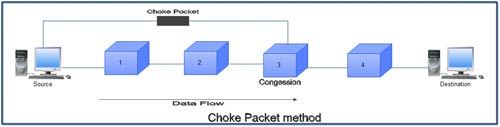
• As shown in fig node 3 is congested and it stops receiving packets and informs its upstream node 2 to slow down. Node 2 in turns may be congested and informs node 1 to slow down. Now node 1 may create congestion and informs the source node to slow down. In this way the congestion is alleviated. Thus, the pressure on node 3 is moved backward to the source to remove the congestion.

### Choke Packet

• In this method of congestion control, congested router or node sends a special type of packet called choke packet to the source to inform it about the congestion.

• Here, congested node does not inform its upstream node about the congestion as in backpressure method.

• In choke packet method, congested node sends a warning directly to the source station *i.e.*the intermediate nodes through which the packet has traveled are not warned.



### Implicit Signaling

• In implicit signaling, there is no communication between the congested node or nodes and the source.

• The source guesses that there is congestion somewhere in the network when it does not receive any acknowledgment. Therefore the delay in receiving an acknowledgment is interpreted as congestion in the network.

• On sensing this congestion, the source slows down.

• This type of congestion control policy is used by TCP.

### Explicit Signaling

• In this method, the congested nodes explicitly send a signal to the source or destination to inform about the congestion.

• Explicit signaling is different from the choke packet method. In choke packed method, a separate packet is used for this purpose whereas in explicit signaling method, the signal is included in the packets that carry data .

• Explicit signaling can occur in either the forward direction or the backward direction .

• In backward signaling, a bit is set in a packet moving in the direction opposite to the congestion. This bit warns the source about the congestion and informs the source to slow down.

• In forward signaling, a bit is set in a packet moving in the direction of congestion. This bit warns the destination about the congestion. The receiver in this case uses policies such as slowing down the acknowledgements to remove the congestion.

## Congestion control algorithms

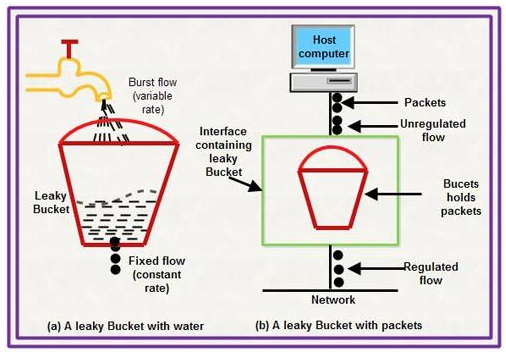
## Leaky Bucket Algorithm

• It is a traffic shaping mechanism that controls the amount and the rate of the traffic sent to the network.

• A leaky bucket algorithm shapes bursty traffic into fixed rate traffic by averaging the data rate.

Imagine a bucket with a small hole at the bottom.

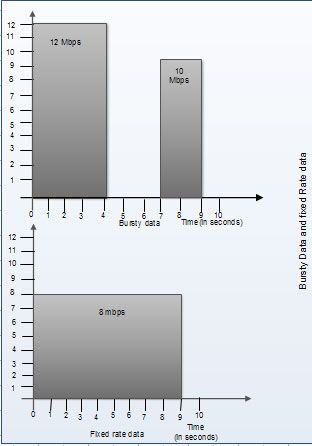
• The rate at which the water is poured into the bucket is not fixed and can vary but it leaks from the bucket at a constant rate. Thus (as long as water is present in bucket), the rate at which the water leaks does not depend on the rate at which the water is input to the bucket



• Also, when the bucket is full, any additional water that enters into the bucket spills over the sides and is lost.

• The same concept can be applied to packets in the network. Consider that data is coming from the source at variable speeds. Suppose that a source sends data at 12 Mbps for 4 seconds. Then there is no data for 3 seconds. The source again transmits data at a rate of 10 Mbps for 2 seconds. Thus, in a time span of 9 seconds, 68 Mb data has been transmitted.

If a leaky bucket algorithm is used, the data flow will be 8 Mbps for 9 seconds. Thus constant flow is maintained.

[](http://ecomputernotes.com/images/Bursty-data-and-fixed-rate-data.jpg)

## Token bucket Algorithm

• The leaky bucket algorithm allows only an average (constant) rate of data flow. Its major problem is that it cannot deal with bursty data.

• A leaky bucket algorithm does not consider the idle time of the host. For example, if the host was idle for 10 seconds and now it is willing to sent data at a very high speed for another 10 seconds, the total data transmission will be divided into 20 seconds and average data rate will be maintained. The host is having no advantage of sitting idle for 10 seconds.

• To overcome this problem, a token bucket algorithm is used. A token bucket algorithm allows bursty data transfers.

• A token bucket algorithm is a modification of leaky bucket in which leaky bucket contains tokens.

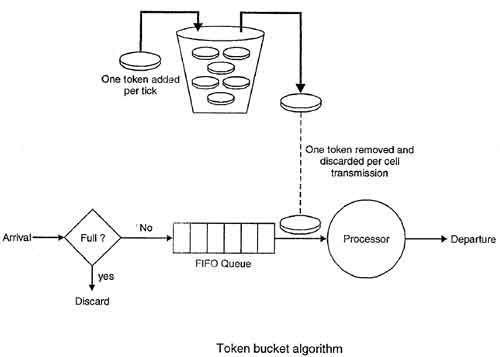
• In this algorithm, a token(s) are generated at every clock tick. For a packet to be transmitted, system must remove token(s) from the bucket.

• Thus, a token bucket algorithm allows idle hosts to accumulate credit for the future in form of tokens.

• For example, if a system generates 100 tokens in one clock tick and the host is idle for 100 ticks. The bucket will contain 10,000 tokens.

Now, if the host wants to send bursty data, it can consume all 10,000 tokens at once for sending 10,000 cells or bytes.

Thus a host can send bursty data as long as bucket is not empty.

[](http://ecomputernotes.com/images/Token-bucket-Algorithm.jpg)

**EX. NO 11 Perform a case study about the different routing algorithms to select the network path with its optimum and economical during data transfer.**

1. **Link State Routing**

Link state routing is the second family of routing protocols. While distance vector routers use a distributed algorithm to compute their routing tables, link-state routing uses link-state routers to exchange messages that allow each router to learn the entire network topology. Based on this learned topology, each router is then able to compute its routing table by using a shortest path computation.

**Features of link state routing protocols**

* **Link state packet –** A small packet that contains routing information.
* **Link state database –** A collection information gathered from link state packet.
* **Shortest path first algorithm (Dijkstra algorithm) –** A calculation performed on the database results into shortest path
* **Routing table –** A list of known paths and interfaces.

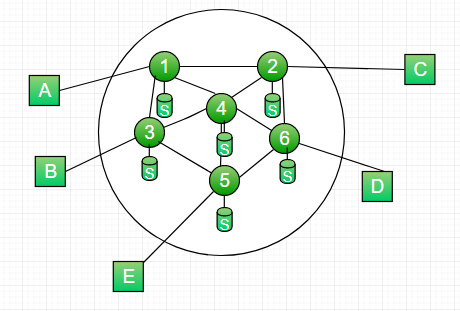
**Calculation of shortest path**

To find shortest path, each node need to run the famous **Dijkstra algorithm**. This famous algorithm uses the following steps:

* **Step-1:** The node is taken and chosen as a root node of the tree, this creates the tree with a single node, and now set the total cost of each node to some value based on the information in Link State Database
* **Step-2:** Now the node selects one node, among all the nodes not in the tree like structure, which is nearest to the root, and adds this to the tree.The shape of the tree gets changed .
* **Step-3:** After this node is added to the tree, the cost of all the nodes not in the tree needs to be updated because the paths may have been changed.
* **Step-4:** The node repeats the Step 2. and Step 3. until all the nodes are added in the tree

Link State protocols in comparison to Distance Vector protocols have:

1. It requires large amount of memory.
2. Shortest path computations require many CPU circles.
3. If network use the little bandwidth ; it quickly reacts to topology changes
4. All items in the database must be sent to neighbors to form link state packets.
5. All neighbors must be trusted in the topology.
6. Authentication mechanisms can be used to avoid undesired adjacency and problems.
7. No split horizon techniques are possible in the link state routing.
8. **Flooding**



**Figure –** A simple packet switching network with six nodes (routers)

* Requires no network information like topology, load condition ,cost of diff. paths
* Every incoming packet to a node is sent out on every outgoing like except the one it arrived on.
* For Example in above figure
  + A incoming packet to (1) is sent out to (2),(3)
  + from (2) is sent to (6),(4) and from (3) it is sent to (4),(5)
  + from (4) it is sent to (6),(5),(3) , from (6) it is sent to (2),(4),(5),from (5) it is sent to (4),(3)

Characteristics

* All possible routes between Source and Destination is tried. A packet will always get through if path exists
* As all routes are tried, the will be atleast one route which is the shortes
* All nodes directly or indirectly connected are visited

Limitations

* Flooding generates vast number of duplicate pakects
* Suitable damping mechanism must be used

Hop-Count

* A hop counter may be contained in the packet header which is decremented at each hop.  
  with the packet being discarded when the counter becomes zero
* The sender intializes the hop counter . If no estimate is known, it is set to the full diameter of the subnet.
* Keep track of the packets which are responsible for flooding using a sequence number . Avoid sending them out a second time.

Selective Flooding : Routers do not send every incoming packet out on every line , only on those lines that go in approximately in the direction of destination.

Advantages of Flooding :

* Highly Robust, emergency or immediate messages can be sent (eg military applications)
* Set up route in virtual circuit
* Flooding always chooses the shortest path
* Broadcast messages to all the nodes

1. **Distance Vector Routing**

A **distance-vector routing (DVR)** protocol requires that a router inform its neighbors of topology changes periodically. Historically known as the old ARPANET routing algorithm (or known as Bellman-Ford algorithm).

**Bellman Ford Basics –** Each router maintains a Distance Vector table containing the distance between itself and ALL possible destination nodes. Distances,based on a chosen metric, are computed using information from the neighbors’ distance vectors.

Information kept by DV router

* Each router has an ID
* Associated with each link connected to a router,
* there is a link cost (static or dynamic).
* Intermediate hops

Distance Vector Table Initialization

* Distance to itself = 0
* Distance to ALL other routers = infinity number.

**Distance Vector Algorithm**

1. A router transmits its distance vector to each of its neighbors in a routing packet.
2. Each router receives and saves the most recently received distance vector from each of its neighbors.
3. A router recalculates its distance vector when:
   * It receives a distance vector from a neighbor containing different information than before.
   * It discovers that a link to a neighbor has gone down.

The DV calculation is based on minimizing the cost to each destination

Dx(y) = Estimate of least cost from x to y

C(x,v) = Node x knows cost to each neighbor v

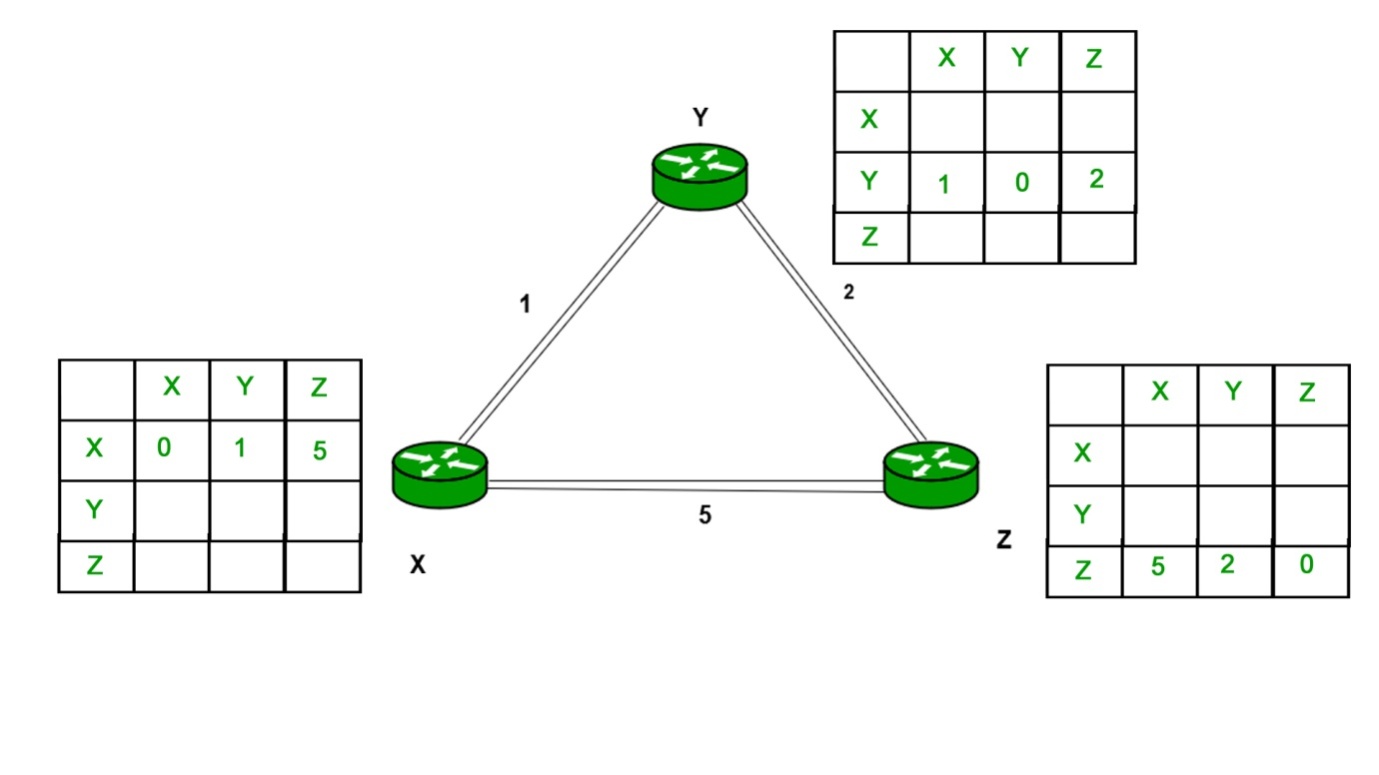
Dx = [Dx(y): y ∈ N ] = Node x maintains distance vector

Node x also maintains its neighbors' distance vectors

– For each neighbor v, x maintains Dv = [Dv(y): y ∈ N ]

* From time-to-time, each node sends its own distance vector estimate to neighbors.
* When a node x receives new DV estimate from any neighbor v, it saves v’s distance vector and it updates its own DV using B-F equation:
* Dx(y) = min { C(x,v) + Dv(y)} for each node y ∈ N

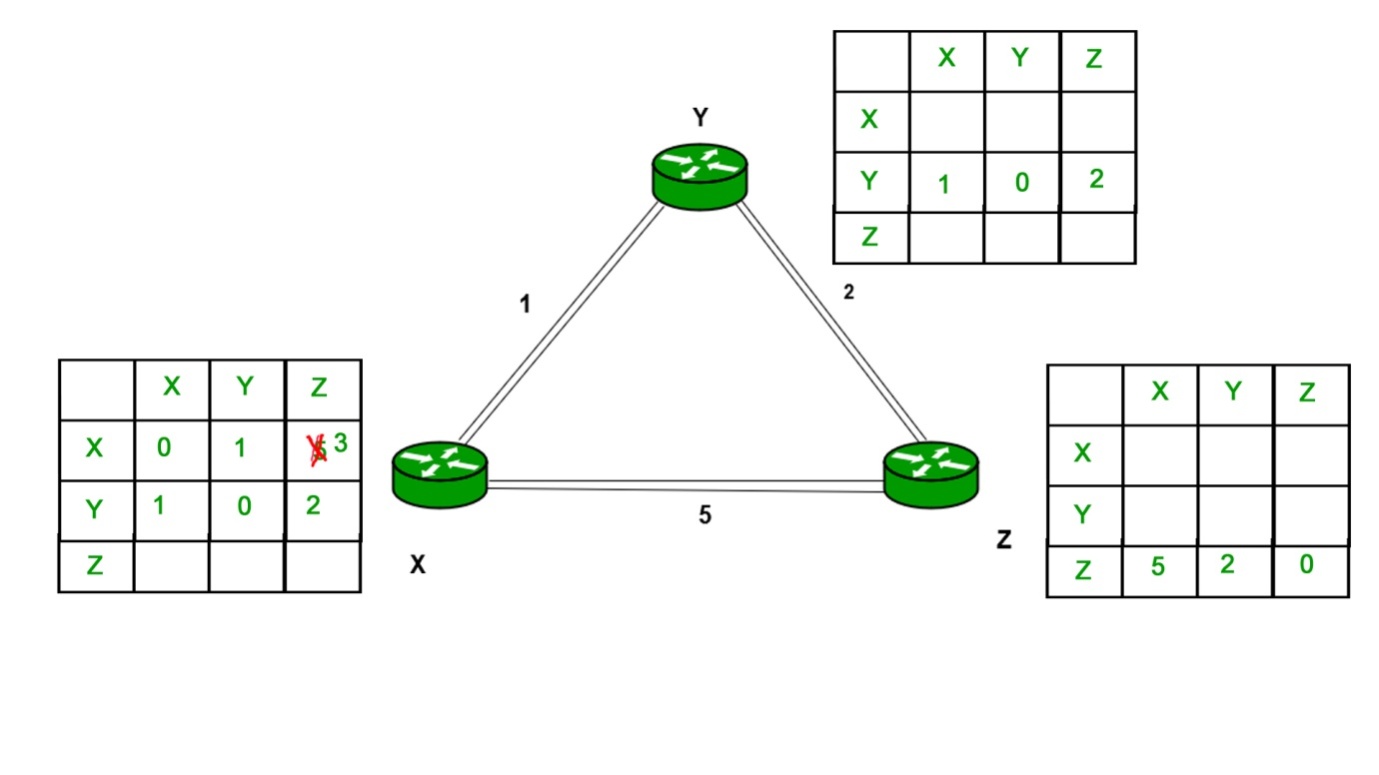
**Example –** Consider 3-routers X, Y and Z as shown in figure. Each router have their routing table. Every routing table will contain distance to the destination nodes.



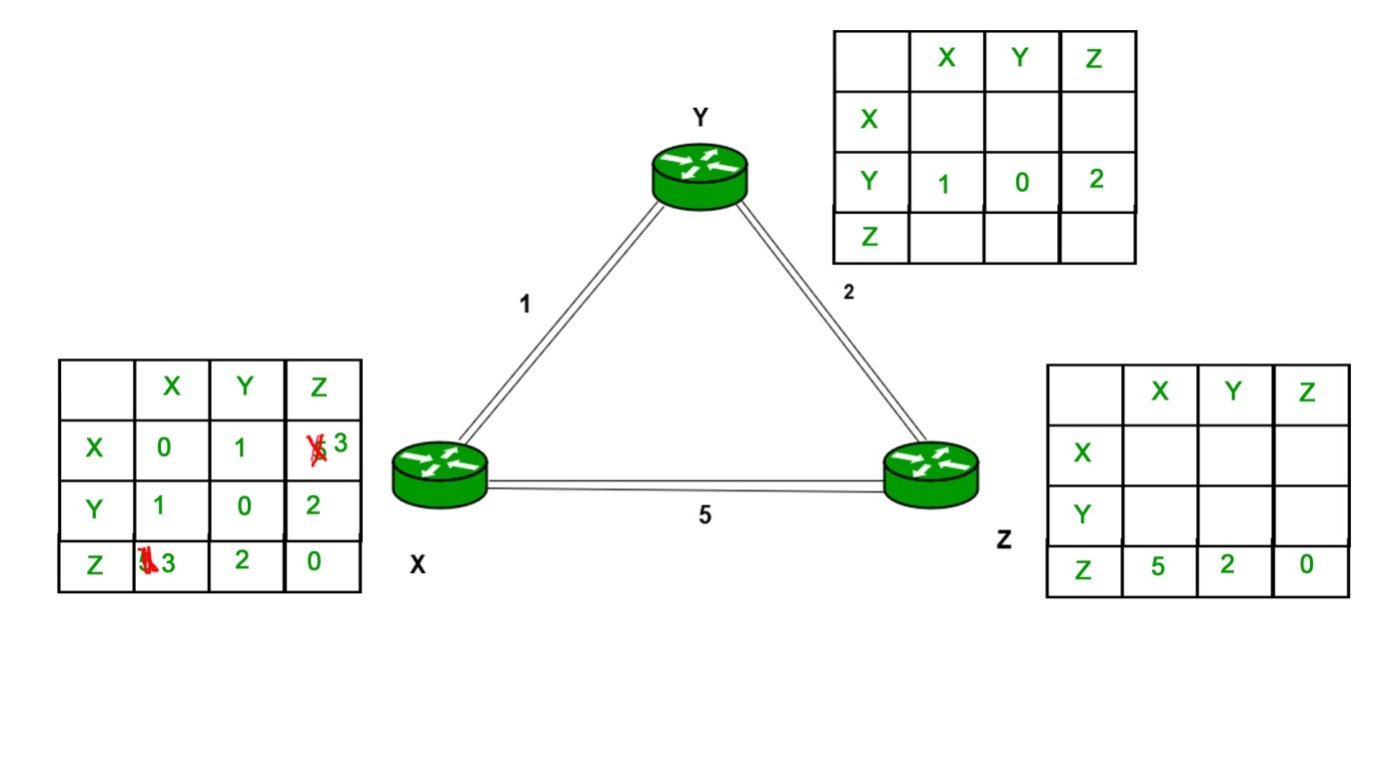
Consider router X , X will share it routing table to neighbors and neighbors will share it routing table to it to X and distance from node X to destination will be calculated using bellmen- ford equation.

Dx(y) = min { C(x,v) + Dv(y)} for each node y ∈ N

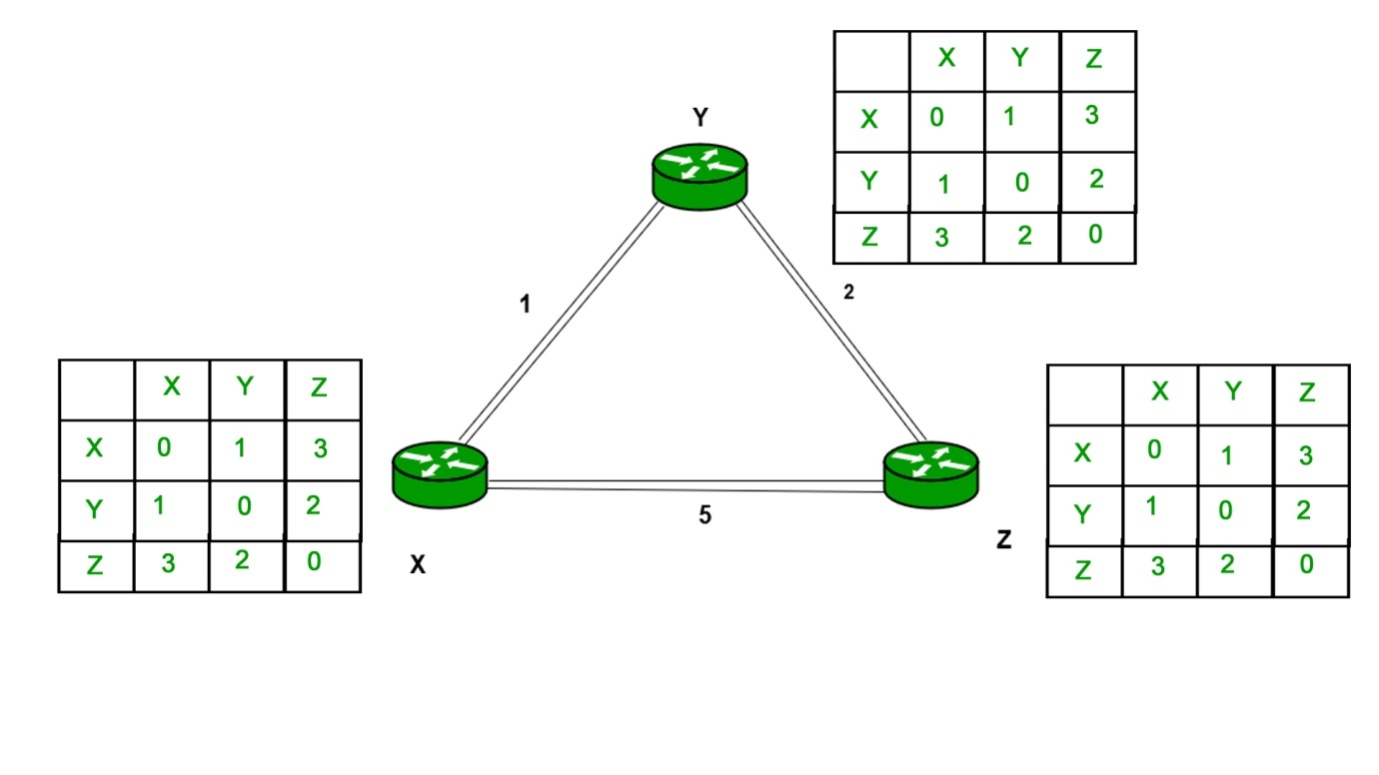
As we can see that distance will be less going from X to Z when Y is intermediate node(hop) so it will be update in routing table X.



Similarly for Z also



Finally the routing table for all



**Advantages of Distance Vector routing**

* It is simpler to configure and maintain than link state routing.

**Disadvantages of Distance Vector routing**

* + It is slower to converge than link state.
  + It is at risk from the count-to-infinity problem.
  + It creates more traffic than link state since a hop count change must be propagated to all routers and processed on each router. Hop count updates take place on a periodic basis, even if there are no changes in the network topology, so bandwidth-wasting broadcasts still occur.
  + For larger networks, distance vector routing results in larger routing tables than link state since each router must know about all other routers. This can also lead to congestion on WAN links.

Distance Vector routing uses UDP(User datagram protocol) for transportation.