Module 2

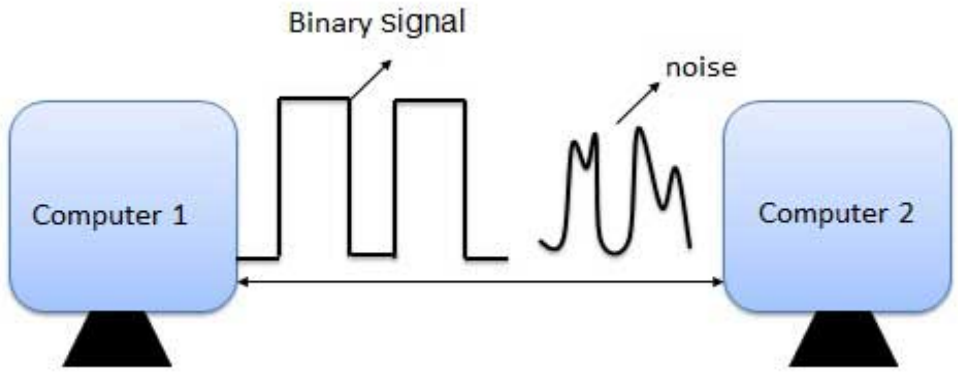
Aim : Implementation of Error Detection / Error Correction Techniques

Theory:

Implementation of Error Detection/Error Correction

What is Error?

Error is a condition when the output information does not match with the input information. During transmission, digital signals suffer from noise that can introduce errors in the binary bits travelling from one system to other. That means a 0 bit may change to 1 or a 1 bit may change to 0.



Error-Detecting codes

Whenever a message is transmitted, it may get scrambled by noise or data may get corrupted. To avoid this, we use error-detecting codes which are additional data added to a given digital message to help us detect if an error occurred during transmission of the message. A simple example of error-detecting code is parity check.

Error-Correcting codes

Along with error-detecting code, we can also pass some data to figure out the original message from the corrupt message that we received. This type of code is called an error-correcting code. Error-correcting codes also deploy the same strategy as error-detecting codes but additionally, such codes also detect the exact location of the corrupt bit.

In error-correcting codes, parity check has a simple way to detect errors along with a sophisticated mechanism to determine the corrupt bit location. Once the corrupt bit is located, its value is reverted (from 0 to 1 or 1 to 0) to get the original message.

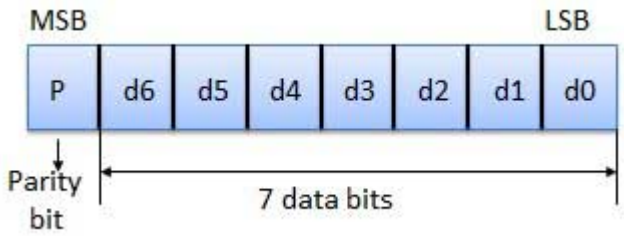
How to Detect and Correct Errors?

To detect and correct the errors, additional bits are added to the data bits at the time of transmission.

- The additional bits are called parity bits. They allow detection or correction of the errors.
- The data bits along with the parity bits form a code word.

Parity Checking of Error Detection

It is the simplest technique for detecting and correcting errors. The MSB of an 8-bits word is used as the parity bit and the remaining 7 bits are used as data or message bits. The parity of 8-bits transmitted word can be either even parity or odd parity.



Even parity -- Even parity means the number of 1's in the given word including the parity bit should be even (2,4,6,....).

Odd parity -- Odd parity means the number of 1's in the given word including the parity bit should be odd (1,3,5,....).

Use of Parity Bit

The parity bit can be set to 0 and 1 depending on the type of the parity required.

- For even parity, this bit is set to 1 or 0 such that the no. of "1 bits" in the entire word is even. Shown in fig. (a).
- For odd parity, this bit is set to 1 or 0 such that the no. of "1 bits" in the entire word is odd. Shown in fig. (b).

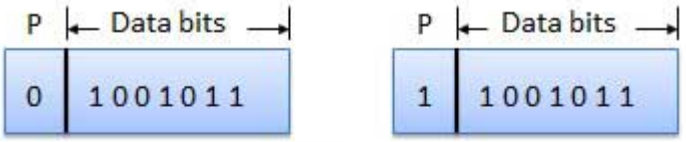


Fig. (a)

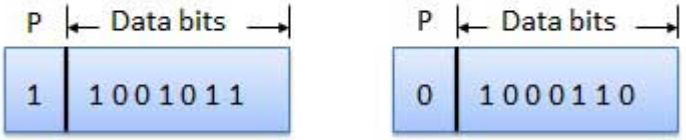
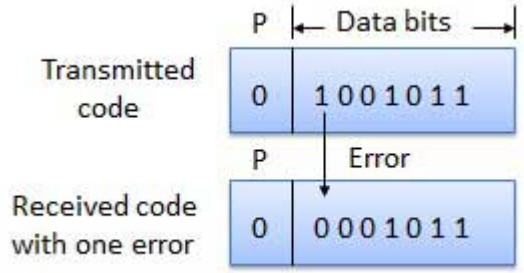


Fig. (b)

How Does Error Detection Take Place?

Parity checking at the receiver can detect the presence of an error if the parity of the receiver signal is different from the expected parity. That means, if it is known that the parity of the transmitted signal is always going to be "even" and if the received signal has an odd parity, then the receiver can conclude that the received signal is not correct. If an error is detected, then the receiver will ignore the received byte and request for retransmission of the same byte to the transmitter.



Program:

```
import java.util.*;


class Hamming {
        public static void main(String args[]) {
                Scanner scan = new Scanner(System.in);
                System.out.println("Enter the number of bits for the Hamming data:");
                int n = scan.nextInt();
                int a[] = new int[n];

                for(int i=0 ; i < n ; i++) {
                        System.out.println("Enter bit no. " + (n-i) + ":");
                        a[n-i-1] = scan.nextInt();
                }

                System.out.println("You entered:");
                for(int i=0 ; i < n ; i++) {
                        System.out.print(a[n-i-1]);
                }
                System.out.println();

                int b[] = generateCode(a);

                System.out.println("Generated code is:");
                for(int i=0 ; i < b.length ; i++) {
                        System.out.print(b[b.length-i-1]);
                }
                System.out.println();
```

```
                // Difference in the sizes of original and new array will give us the number of parity
bits added.

                System.out.println("Enter position of a bit to alter to check for error detection at the
receiver end (0 for no error):");

                int error = scan.nextInt();

                if(error != 0) {

                        b[error-1] = (b[error-1]+1)%2;

                }

                System.out.println("Sent code is:");

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

                        System.out.print(b[b.length-i-1]);

                }

                System.out.println();

                receive(b, b.length - a.length);

        }


        static int[] generateCode(int a[]) {

                // We will return the array 'b'.

                int b[];


                // We find the number of parity bits required:

                int i=0, parity_count=0 ,j=0, k=0;

                while(i < a.length) {

                        // 2^(parity bits) must equal the current position

                        // Current position is (number of bits traversed + number of parity bits + 1).

                        // +1 is needed since array indices start from 0 whereas we need to start from
1.


                        if(Math.pow(2,parity_count) == i+parity_count + 1) {

                                parity_count++;
```

```
            }
            else {
                        i++;
            }
}

// Length of 'b' is length of original data (a) + number of parity bits.
b = new int[a.length + parity_count];

// Initialize this array with '2' to indicate an 'unset' value in parity bit locations:

for(i=1 ; i <= b.length ; i++) {
            if(Math.pow(2, j) == i) {
            // Found a parity bit location.
            // Adjusting with (-1) to account for array indices starting from 0 instead of 1.

                        b[i-1] = 2;
                        j++;
            }
            else {
                        b[k+j] = a[k++];
            }
}
for(i=0 ; i < parity_count ; i++) {
            // Setting even parity bits at parity bit locations:

            b[((int) Math.pow(2, i))-1] = getParity(b, i);
}
return b;
```

```
	}

	static int getParity(int b[], int power) {
		int parity = 0;
		for(int i=0 ; i < b.length ; i++) {
			if(b[i] != 2) {
				// If 'i' doesn't contain an unset value,
				// We will save that index value in k, increase it by 1,
				// Then we convert it into binary:

				int k = i+1;
				String s = Integer.toBinaryString(k);

				//Nw if the bit at the 2^(power) location of the binary value of index is
1
				//Then we need to check the value stored at that location.
				//Checking if that value is 1 or 0, we will calculate the parity value.

				int x = ((Integer.parseInt(s))/((int) Math.pow(10, power)))%10;
				if(x == 1) {
					if(b[i] == 1) {
						parity = (parity+1)%2;
					}
				}
			}
		}
		return parity;
	}
```

```
static void receive(int a[], int parity_count) {
        // This is the receiver code. It receives a Hamming code in array 'a'.
        // We also require the number of parity bits added to the original data.
        // Now it must detect the error and correct it, if any.

        int power;
        // We shall use the value stored in 'power' to find the correct bits to check for parity.

        int parity[] = new int[parity_count];
        // 'parity' array will store the values of the parity checks.

        String syndrome = new String();
        // 'syndrome' string will be used to store the integer value of error location.

        for(power=0 ; power < parity_count ; power++) {
        // We need to check the parities, the same no of times as the no of parity bits added.

                for(int i=0 ; i < a.length ; i++) {
                        // Extracting the bit from 2^(power):

                        int k = i+1;
                        String s = Integer.toBinaryString(k);
                        int bit = ((Integer.parseInt(s))/((int) Math.pow(10, power)))%10;
                        if(bit == 1) {
                                if(a[i] == 1) {
                                        parity[power] = (parity[power]+1)%2;
                                }
                        }
                }
```

```
                syndrome = parity[power] + syndrome;
        }
        // This gives us the parity check equation values.
        // Using these values, we will now check if there is a single bit error and then correct
it.

        int error_location = Integer.parseInt(syndrome, 2);
        if(error_location != 0) {
                System.out.println("Error is at location " + error_location + ".");
                a[error_location-1] = (a[error_location-1]+1)%2;
                System.out.println("Corrected code is:");
                for(int i=0 ; i < a.length ; i++) {
                        System.out.print(a[a.length-i-1]);
                }
                System.out.println();
        }
        else {
                System.out.println("There is no error in the received data.");
        }

        // Finally, we shall extract the original data from the received (and corrected) code:
        System.out.println("Original data sent was:");
        power = parity_count-1;
        for(int i=a.length ; i > 0 ; i--) {
                if(Math.pow(2, power) != i) {
                        System.out.print(a[i-1]);
                }
                else {
                        power--;
```

```
                }
            }
            System.out.println();
        }
}
```

OUTPUT :

Aim: Implementation of Stop and Wait Protocol and sliding window

Theory:

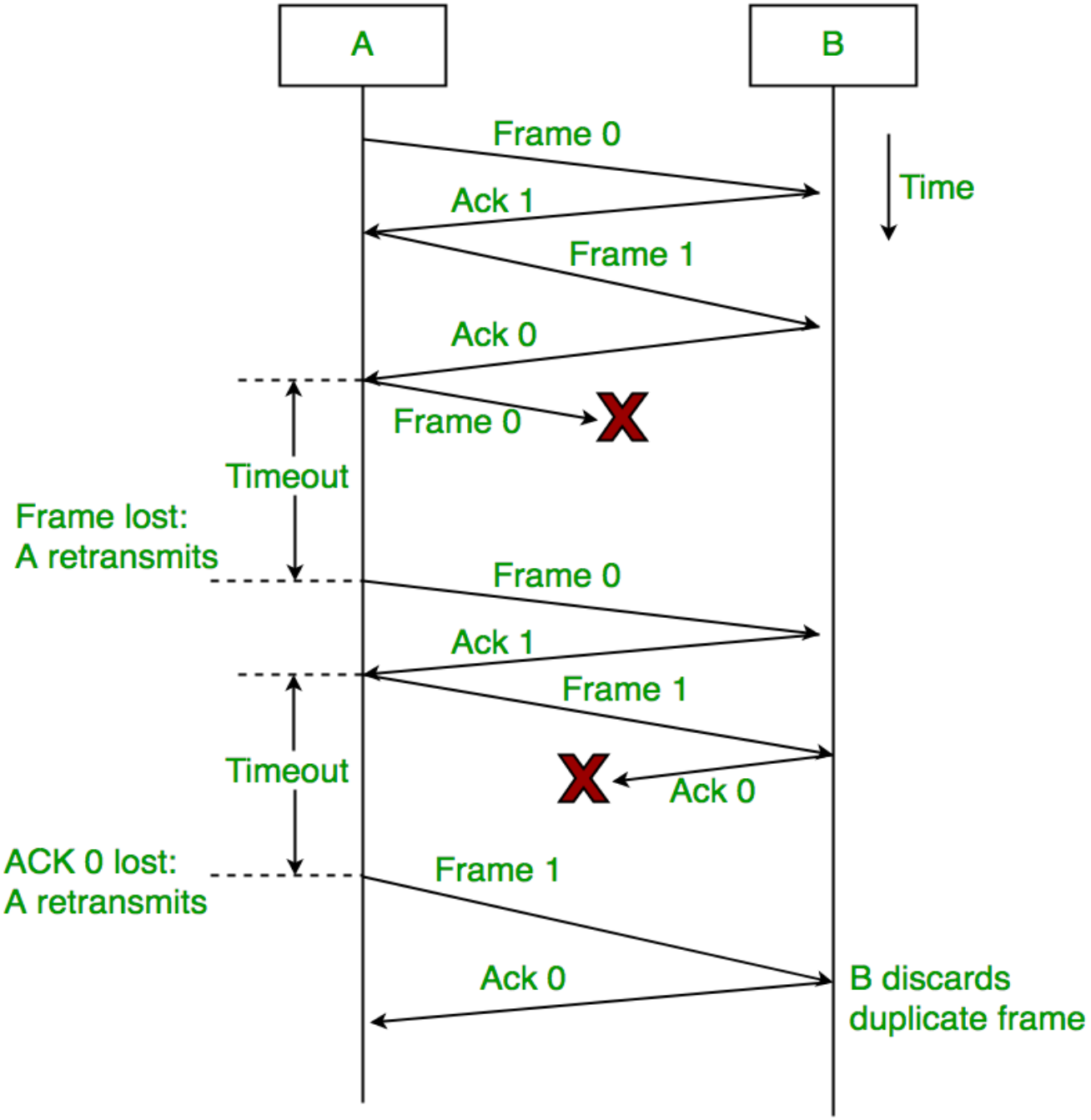
Implementation of stop and wait protocol

Working of stop and wait for ARQ:

1) Sender A sends a data frame or packet with sequence number 0.

2) Receiver B, after receiving the data frame, sends an acknowledgement with sequence number 1 (the sequence number of the next expected data frame or packet)

There is only a one-bit sequence number that implies that both sender and receiver have a buffer for one frame or packet only.



Characteristics of Stop and Wait ARQ:

It uses a link between sender and receiver as a half-duplex link

Throughput = 1 Data packet/frame per RTT

If the Bandwidth*Delay product is very high, then they stop and wait for protocol if it is not so useful. The sender has to keep waiting for acknowledgements before sending the processed next packet.

It is an example of “Closed Loop OR connection-oriented “ protocols

It is a special category of SWP where its window size is 1

Irrespective of the number of packets sender is having stop and wait for protocol requires only 2 sequence numbers 0 and 1

The Stop and Wait ARQ solves the main three problems but may cause big performance issues as the sender always waits for acknowledgement even if it has the next packet ready to send. Consider a situation where you have a high bandwidth connection and propagation delay is also high (you are connected to some server in some other country through a high-speed connection). To solve this problem, we can send more than one packet at a time with a larger sequence number. We will be discussing these protocols in the next articles.

So Stop and Wait ARQ may work fine where propagation delay is very less for example LAN connections but performs badly for distant connections like satellite connections.

Program:

//Sender Program

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

public class Sender {
   Socket sender;
   ObjectOutputStream out;
   ObjectInputStream in;
   String packet, ack, str, msg;
   int n, i = 0, sequence = 0;

   Sender() {
   }

   public void run() {
      try {
         BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
         System.out.println("Waiting for Connection....");
         sender = new Socket("localhost", 2004);
         sequence = 0;

         out = new ObjectOutputStream(sender.getOutputStream());
         out.flush();
         in = new ObjectInputStream(sender.getInputStream());
```

```
            str = (String) in.readObject();
            System.out.println("reciver     > " + str);
            System.out.println("Enter the data to send....");
            packet = br.readLine();
            n = packet.length();
            do {
               try {
                  if (i < n) {
                     msg = String.valueOf(sequence);
                     msg = msg.concat(packet.substring(i, i + 1));
                  } else if (i == n) {
                     msg = "end";
                     out.writeObject(msg);
                     break;
                  }
                  out.writeObject(msg);
                  sequence = (sequence == 0) ? 1 : 0;
                  out.flush();
                  System.out.println("data sent>" + msg);
                  ack = (String) in.readObject();
                  System.out.println("waiting for ack.....\n\n");
                  if (ack.equals(String.valueOf(sequence))) {
                     i++;
                     System.out.println("receiver   >  " + " packet recieved\n\n");
                  } else {
                     System.out.println("Time out resending data....\n\n");
                     sequence = (sequence == 0) ? 1 : 0;
                  }
               } catch (Exception e) {
               }
            } while (i < n + 1);
            System.out.println("All data sent. exiting.");
         } catch (Exception e) {
         } finally {
            try {
               in.close();
               out.close();
               sender.close();
            } catch (Exception e) {
            }
         }
      }

      public static void main(String args[]) {
         Sender s = new Sender();
         s.run();
      }
   }
```

//Receiver Program

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

public class Reciever {
   ServerSocket reciever;
   Socket connection = null;
   ObjectOutputStream out;
   ObjectInputStream in;
   String packet, ack, data = "";
   int i = 0, sequence = 0;

   Reciever() {
   }

   public void run() {
      try {
         BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
         reciever = new ServerSocket(2004, 10);
         System.out.println("waiting for connection...");
         connection = reciever.accept();
         sequence = 0;
         System.out.println("Connection established   :");
         out = new ObjectOutputStream(connection.getOutputStream());
         out.flush();
         in = new ObjectInputStream(connection.getInputStream());
         out.writeObject("connected    .");
         do {
            try {
               packet = (String) in.readObject();
               if (Integer.valueOf(packet.substring(0, 1)) == sequence) {
                  data += packet.substring(1);
                  sequence = (sequence == 0) ? 1 : 0;
                  System.out.println("\n\nreceiver        >" + packet);
               } else {
                  System.out.println("\n\nreceiver        >" + packet + "   duplicate
data");
               }
               if (i < 3) {
                  out.writeObject(String.valueOf(sequence));
                  i++;
               } else {
                  out.writeObject(String.valueOf((sequence + 1) % 2));
```

```
                    i = 0;
                }
            } catch (Exception e) {
            }
        } while (!packet.equals("end"));
        System.out.println("Data recived=" + data);
        out.writeObject("connection ended    .");
    } catch (Exception e) {
    } finally {
        try {
            in.close();
            out.close();
            reciever.close();
        } catch (Exception e) {
        }
    }
  }

  public static void main(String args[]) {
    Reciever s = new Reciever();
    while (true) {
        s.run();
    }
  }
}
```

OUTPUT : {SENDER and Receiver}

AIM: Implementation of sliding window Protocol

Theory:

Sliding Window Protocol

The sliding window is a technique for sending multiple frames at a time. It controls the data packets between the two devices where reliable and gradual delivery of data frames is needed. It is also used in TCP (Transmission Control Protocol).

In this technique, each frame has sent from the sequence number. The sequence numbers are used to find the missing data in the receiver end. The purpose of the sliding window technique is to avoid duplicate data, so it uses the sequence number.

Program:

//SENDER PROGRAM

```
 import java.net.*;

import java.io.*;

import java.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 = new DataInputStream (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 the no. 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 = new DataInputStream (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 ("\nAcknowledgment sent\n");

p.println (rptr + 1);
            rws += nf;
          }

        else

break;

ch = in.readLine ();

}

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

}

}
```

OUTPUT : {SENDER and Receiver}

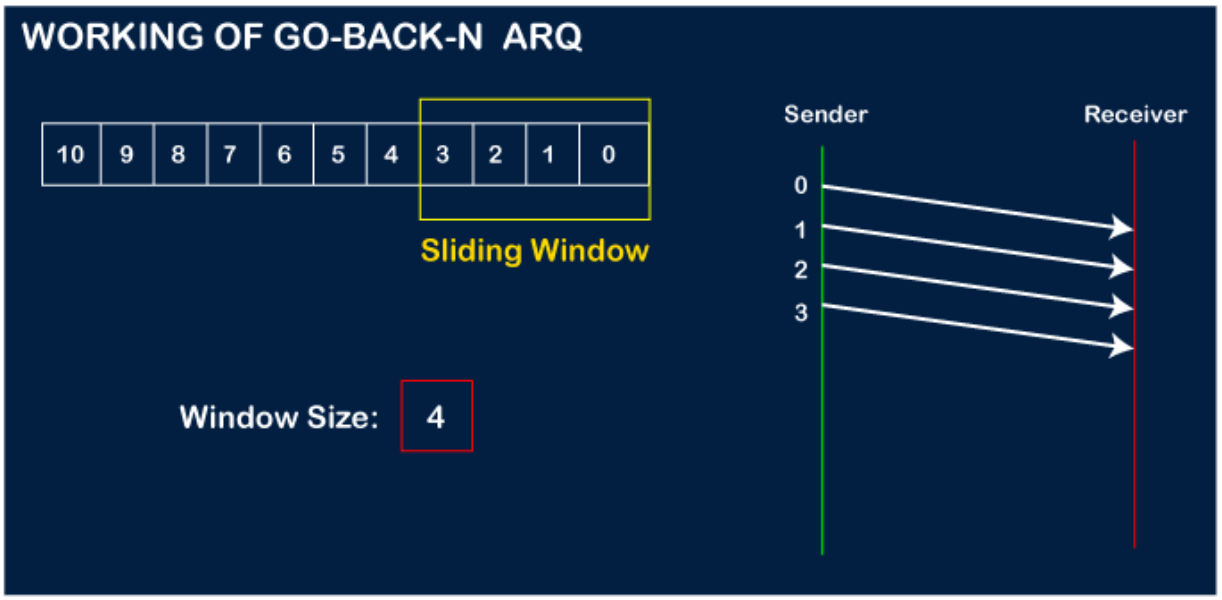
Aim : Implementation and study of Goback-N ARQ

Theory : Go-Back-N ARQ

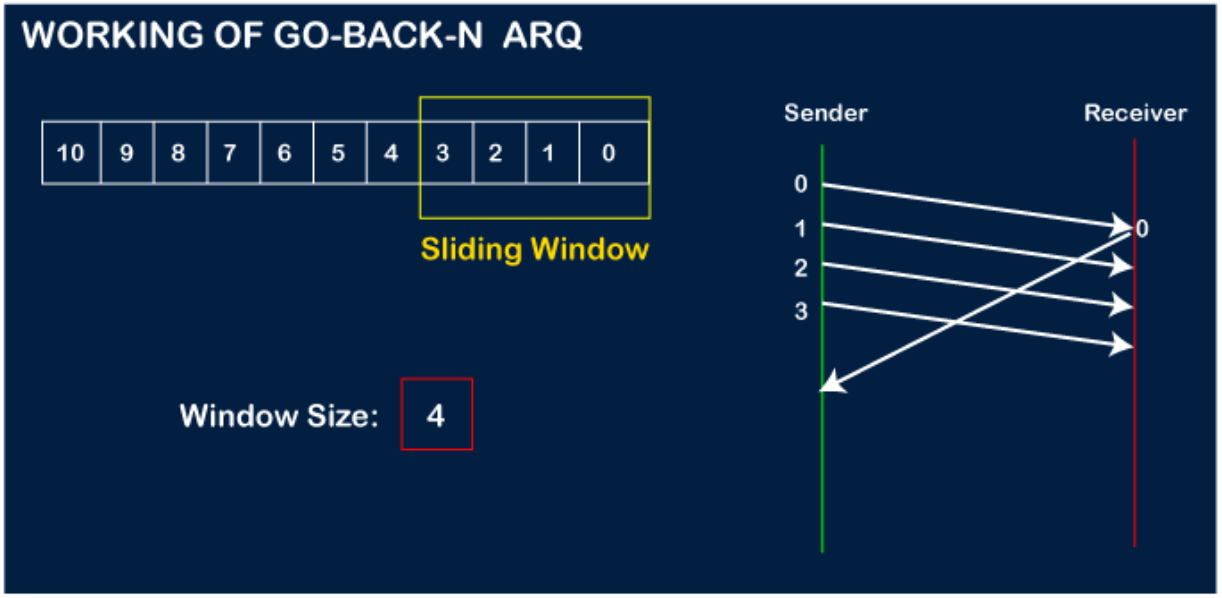
Working of Go-Back-N ARQ

Consider a sender and a receiver, and let's assume that there are 11 frames to be sent. These frames are represented as 0,1,2,3,4,5,6,7,8,9,10, and these are the sequence numbers of the frames. Mainly, the sequence number is decided by the sender's window size. But, for better understanding, we took the running sequence numbers, i.e., 0,1,2,3,4,5,6,7,8,9,10. Consider the window size as 4, which means that the four frames can be sent at a time before expecting the acknowledgment of the first frame.

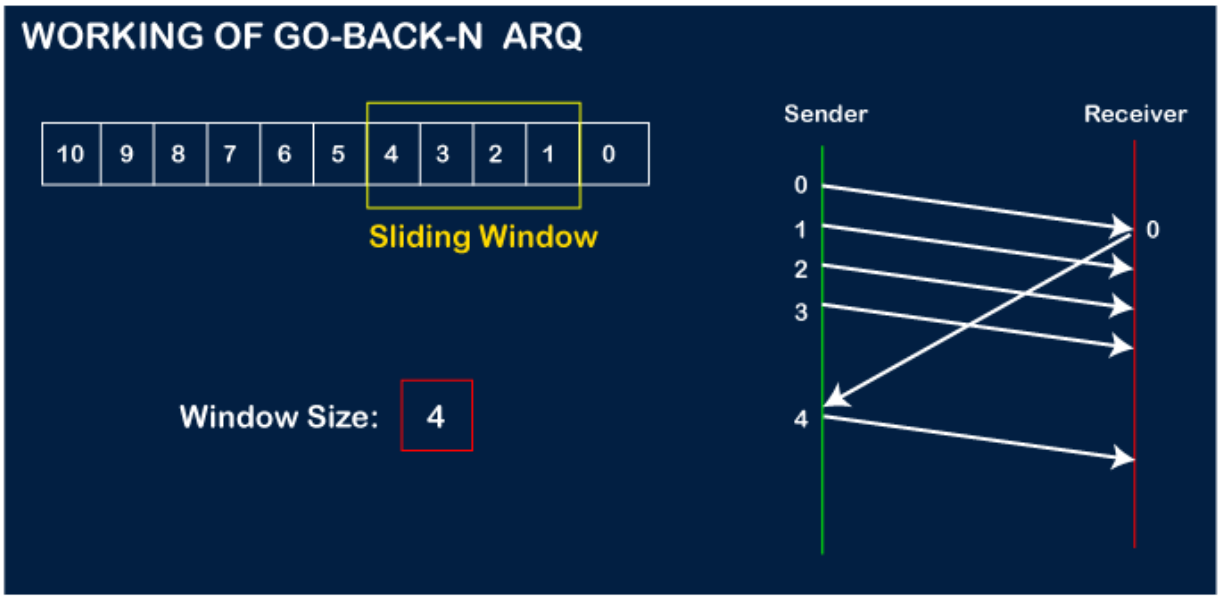
Step 1: Initially, the sender will send the first four frames to the receiver, i.e., 0,1,2,3, and now the sender is expected to receive the acknowledgment of the 0^{th} frame.



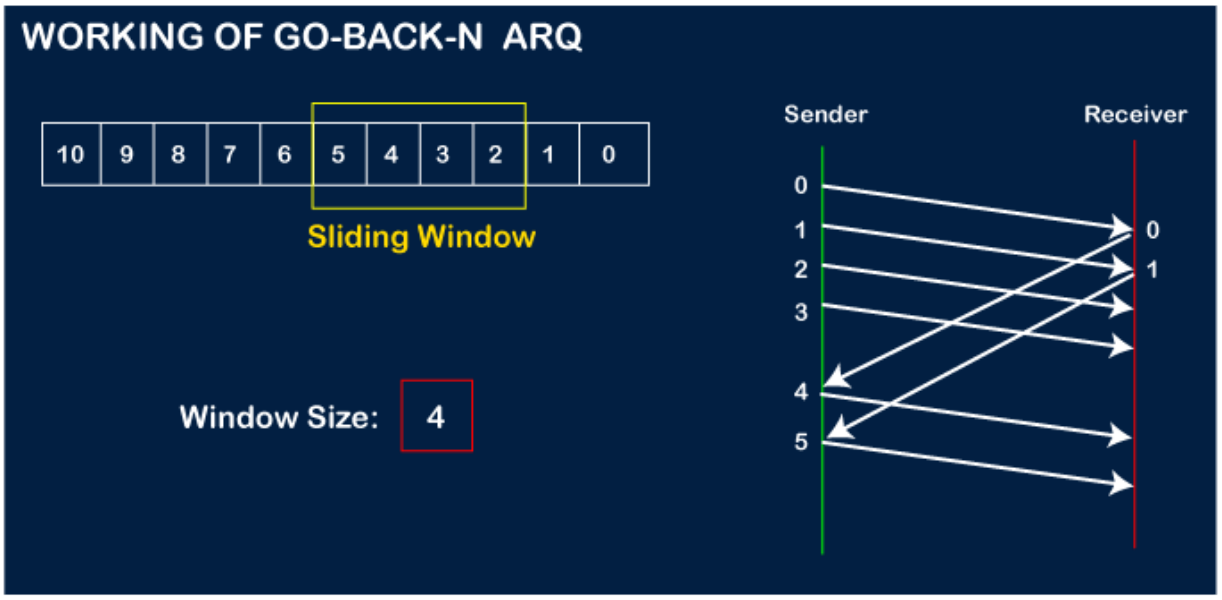
Assume that the receiver has sent the acknowledgment for the 0 frame, and the receiver has successfully received it.



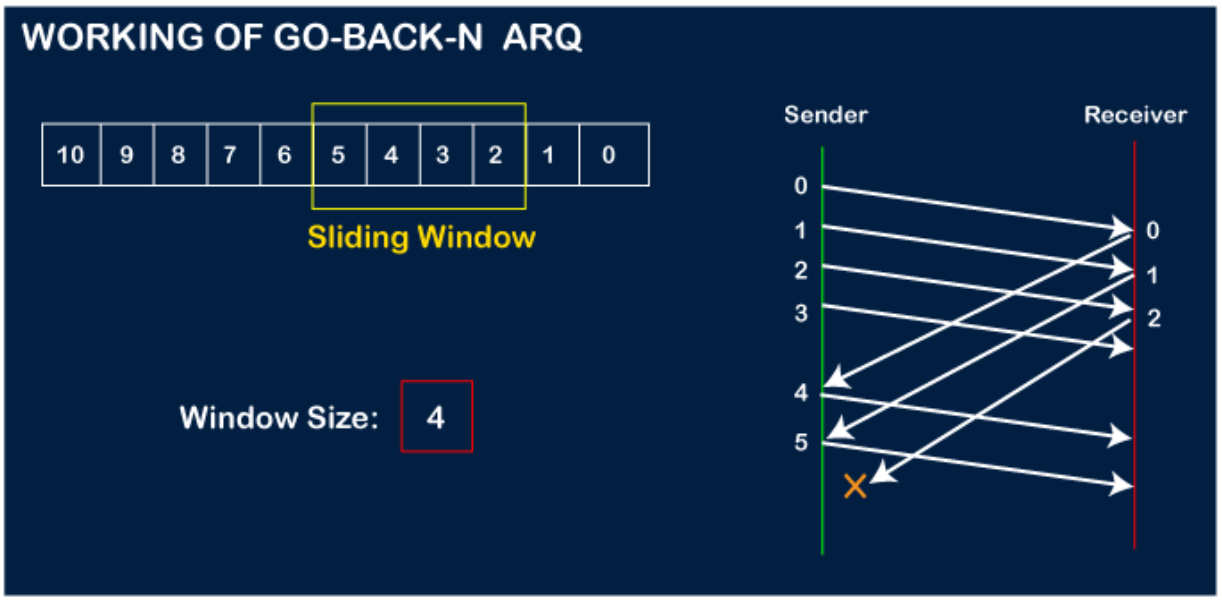
The sender will then send the next frame, i.e., 4, and the window slides containing four frames (1,2,3,4).

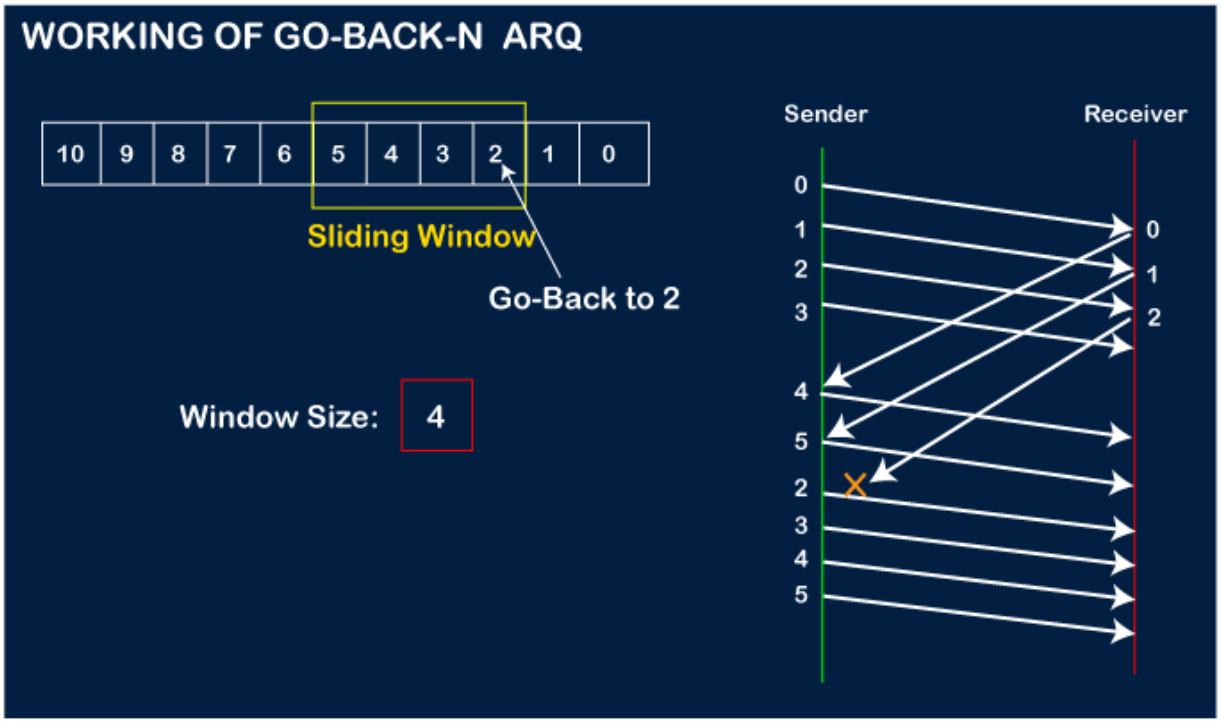


The receiver will then send the acknowledgement for frame no 1. After receiving the acknowledgment, the sender will send the next frame, i.e., frame no 5, and the window will slide having four frames (2,3,4,5).



Now, The receiver is not acknowledging the frame no 2, either the frame is lost, or the acknowledgement is lost. Instead of sending the frame no 6, the sender Go-Back to 2, which is the first frame of the current window, retransmits all the frames in the current window, i.e., 2,3,4,5.





Program:

```
import java.io.*;

public class GoBackN {

    public static void main(String args[]) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        System.out.println("Please enter the Window Size: ");
        int window = Integer.parseInt(br.readLine());

        boolean loop = true;
        int sent = 0;
```

```
        while (loop) {

            for (int i = 0; i < window; i++) {
                System.out.println("Frame " + sent + " has been transmitted.");
                sent++;
                if (sent == window)
                    break;
            }

            System.out.println("Please enter the last Acknowledgement received.");
            int ack = Integer.parseInt(br.readLine());

            if (ack == window)
                loop = false;
            else
                sent = ack;
        }

    }

}
```

AIM : Implementation and study of selective repeat protocols

Theory : Selective Repeat

Working of SR

Selective Repeat protocol provides for sending multiple frames depending upon the availability of frames in the sending window, even if it does not receive acknowledgement for any frame in the interim. The maximum number of frames that can be sent depends upon the size of the sending window.

The receiver records the sequence number of the earliest incorrect or un-received frame. It then fills the receiving window with the subsequent frames that it has received. It sends the sequence number of the missing frame along with every acknowledgement frame.

The sender continues to send frames that are in its sending window. Once, it has sent all the frames in the window, it retransmits the frame whose sequence

number is given by the acknowledgements. It then continues sending the other frames.

Program:

//Server Program

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketException;

public class Server {
    static ServerSocket Serversocket;
    static DataInputStream dis;
    static DataOutputStream dos;

    public static void main(String[] args) throws SocketException {

        try {
            int a[] = { 30, 40, 50, 60, 70, 80, 90, 100 };
            Serversocket = new ServerSocket(8011);
            System.out.println("waiting for connection");
            Socket client = Serversocket.accept();
            dis = new DataInputStream(client.getInputStream());
            dos = new DataOutputStream(client.getOutputStream());
            System.out.println("The number of packets sent is:" + a.length);
            int y = a.length;
            dos.write(y);
            dos.flush();

            for (int i = 0; i < a.length; i++) {
                dos.write(a[i]);
                dos.flush();
```

```
            }

            int k = dis.read();

            dos.write(a[k]);
            dos.flush();

        } catch (IOException e) {
            System.out.println(e);
        } finally {
            try {
                dis.close();
                dos.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

//Client Program

```
import java.lang.System;
import java.net.*;
import java.io.*;
import java.text.*;
import java.util.Random;
import java.util.*;

public class Client {
    static Socket connection;

    public static void main(String a[]) throws SocketException {
        try {
            int v[] = new int[10];
            int n = 0;
```

```
Random rands = new Random();
int rand = 0;

InetAddress addr = InetAddress.getByName("Localhost");
System.out.println(addr);
connection = new Socket(addr, 8011);
DataOutputStream out = new DataOutputStream(
        connection.getOutputStream());
DataInputStream in = new DataInputStream(
        connection.getInputStream());
int p = in.read();
System.out.println("No of frame is:" + p);

for (int i = 0; i < p; i++) {
    v[i] = in.read();
    System.out.println(v[i]);
    // g[i] = v[i];
}
rand = rands.nextInt(p);// FRAME NO. IS RANDOMLY GENERATED
v[rand] = -1;
for (int i = 0; i < p; i++) {
    System.out.println("Received frame is: " + v[i]);

}
for (int i = 0; i < p; i++)
    if (v[i] == -1) {
        System.out.println("Request to retransmit from packet no "
                + (i + 1) + " again!!");
        n = i;
        out.write(n);
        out.flush();
    }

System.out.println();
```

```
            v[n] = in.read();
            System.out.println("Received frame is: " + v[n]);

            System.out.println("quiting");
        } catch (Exception e) {
            System.out.println(e);
        }

    }
}
```

OUTPUT : {SERVER and CLIENT}

AIM : Study of Socket Programming and a socket Program for Echo/Ping/Talk commands using java

Theory :

socket programming

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server.

Echo command

In Linux, the echo command can be used for displaying a line of string/text that is passed as the **arguments.** This command is a built-in that is mostly and widely used in various batch files and shell scripts to outcome status test to a file and screen.

Syntax :

```
echo [option] [string]
```

Options	Description
\b	It removes all the spaces in between the text

\c	Suppresses trailing new line with backspace interpreter ‘-e‘ to continue without emitting new line.
\n	This option creates new line from where it is used.
\t	This option is used to create horizontal tab spaces.
\r	Carriage return with backspace interpreter ‘-e‘ to have specified carriage return in output.
\v	This option is used to create vertical tab spaces.
\a	Alert return with backspace interpreter ‘-e‘ to have sound alert.
*****	This command will print all files/folders, similar to ls command.
-n	This option is used to omit echoing trailing newline .

PING command

Ping is a simple, widely used, cross-platform networking utility for testing if a host is reachable on an **Internet Protocol** (**IP**) network. It works by sending a series of **Internet Control Message Protocol** (**ICMP**) **ECHO_REQUEST** messages to the target host and waiting for an **ICMP** echo reply (or **ECHO_RESPONSE**).

The ping command takes the syntax shown.

```
$ ping options IP address
```

Flags	Description
-c	Specifies the number of **ECHO_REQUEST’s** to be sent after which ping exits.

`-i`	Allows you to set interval in seconds between sending each packet, the default value is one second.
`-f`	Determines the response of your network under high-load conditions, you can run a “**flood ping**” which sends requests as fast as possible,
`-b`	Enables pinging a broadcast
`-t`	Limits the number of network hops (**TTL** – **Time-to-live**) that probes traverse.
`-s`	The default packet size should be sufficient for a ping test; however, you can change it to meet your specific testing needs. You can specify the size of the payload.
`-l`	If preload is specified, ping sends that many packets not waiting for reply.
`-W`	It is also possible to set the time to wait for a response, in seconds, using the `-W` option as shown.
`-w`	To set a **timeout** in seconds, before ping exits regardless of how many packets have been sent or received, use the `-w` flag.
`-d`	The `-d` option allows you to enable the debug IP packet detail.
`-v`	enables verbose output using the `-v` flag

Talk command

The /usr/bin/talk command **allows two users on the same host or on different hosts to have an interactive conversation**. The talk command opens both a send window and a receive window on each user's display. Each user is then able to type into the send window while the talk command displays what the other user is typing.

AIM : Implementation of distance vector routing algorithm

Theory :

Distance Vector Routing Algorithm

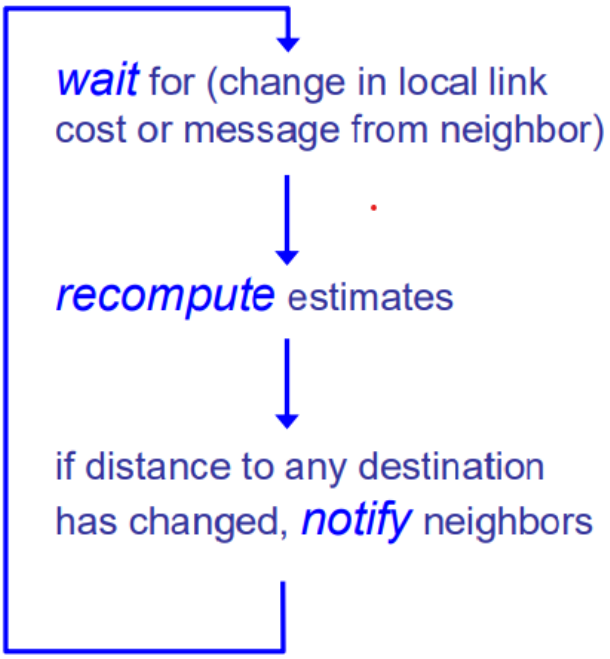
Iterative, asynchronous:

Each local iteration caused by:

- Local link cost change
- Distance vector update message from neighbor

Distributed:

- Each node notifies neighbors only when its DV changes
- Neighbors then notify their neighbors if necessary



Step By Step

- $c(x, v)$ = cost for direct link from x to v
  - ❖ Node x maintains costs of direct links c(x,v)
- $D_x(y)$ = estimate of least cost from x to y
  - ❖ Node x maintains distance vector $D_x = [D_x(y): y \in N]$
- Node x maintains its neighbors' distance vectors
  - ❖ For each neighbor v, x maintains $D_v = [D_v(y): y \in N]$
- Each node v periodically sends D_v to its neighbors
  - ❖ And neighbors update their own distance vectors
  - ❖ $D_x(y) \leftarrow \min_v\{c(x, v) + D_v(y)\}$ for each node $y \in N$

Program :

```
import java.io.*;

public class DVR

{

 static int graph[][];

 static int via[][];
```

```
static int rt[][];
static int v;
static int e;

public static void main(String args[]) throws IOException
{
 BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

 System.out.println("Please enter the number of Vertices: ");
 v = Integer.parseInt(br.readLine());

 System.out.println("Please enter the number of Edges: ");
 e = Integer.parseInt(br.readLine());

 graph = new int[v][v];
 via = new int[v][v];
 rt = new int[v][v];
 for(int i = 0; i < v; i++)
  for(int j = 0; j < v; j++)
  {
   if(i == j)
    graph[i][j] = 0;
   else
    graph[i][j] = 9999;
  }

 for(int i = 0; i < e; i++)
 {
```

```
   System.out.println("Please enter data for Edge " + (i + 1) + ":");
   System.out.print("Source: ");
   int s = Integer.parseInt(br.readLine());
   s--;
   System.out.print("Destination: ");
   int d = Integer.parseInt(br.readLine());
   d--;
   System.out.print("Cost: ");
   int c = Integer.parseInt(br.readLine());
   graph[s][d] = c;
   graph[d][s] = c;
  }

  dvr_calc_disp("The initial Routing Tables are: ");

  System.out.print("Please enter the Source Node for the edge whose cost has
changed: ");
  int s = Integer.parseInt(br.readLine());
  s--;
  System.out.print("Please enter the Destination Node for the edge whose cost has
changed: ");
  int d = Integer.parseInt(br.readLine());
  d--;
  System.out.print("Please enter the new cost: ");
  int c = Integer.parseInt(br.readLine());
  graph[s][d] = c;
  graph[d][s] = c;

  dvr_calc_disp("The new Routing Tables are: ");
```

```
}

static void dvr_calc_disp(String message)
{
 System.out.println();
 init_tables();
 update_tables();
 System.out.println(message);
 print_tables();
 System.out.println();
}

static void update_table(int source)
{
 for(int i = 0; i < v; i++)
 {
  if(graph[source][i] != 9999)
  {
   int dist = graph[source][i];
   for(int j = 0; j < v; j++)
   {
    int inter_dist = rt[i][j];
    if(via[i][j] == source)
     inter_dist = 9999;
    if(dist + inter_dist < rt[source][j])
    {
     rt[source][j] = dist + inter_dist;
     via[source][j] = i;
```

```
    }
   }
  }
 }
}

static void update_tables()
{
 int k = 0;
 for(int i = 0; i < 4*v; i++)
 {
  update_table(k);
  k++;
  if(k == v)
   k = 0;
 }
}

static void init_tables()
{
 for(int i = 0; i < v; i++)
 {
  for(int j = 0; j < v; j++)
  {
   if(i == j)
   {
    rt[i][j] = 0;
    via[i][j] = i;
```

```
    }
    else
    {
     rt[i][j] = 9999;
     via[i][j] = 100;
    }
   }
  }
}

 static void print_tables()
 {
  for(int i = 0; i < v; i++)
  {
   for(int j = 0; j < v; j++)
   {
    System.out.print("Dist: " + rt[i][j] + "     ");
   }
   System.out.println();
  }
 }

}
```

OUTPUT :

AIM : Implementation of Link state routing algorithm

Theory :

Link state Routing Algorithm

Notations:

- **c(i , j):** Link cost from node i to node j. If i and j nodes are not directly linked, then c(i , j) = ∞.
- **D(v):** It defines the cost of the path from source code to destination v that has the least cost currently.
- **P(v):** It defines the previous node (neighbor of v) along with current least cost path from source to v.
- **N:** It is the total number of nodes available in the network.

Algorithm

Initialization

N = {A} // **A is a root node**.
for all nodes v
if v adjacent to A
then D(v) = c(A,v)
else D(v) = infinity
loop
find w not in N such that D(w) is a minimum.
Add w to N
Update D(v) for all v adjacent to w and not in N:
D(v) = min(D(v) , D(w) + c(w,v))
Until all nodes in N

Program:

```
import java.util.*;
public class LSR
{
    public static void main(String[] args)
   {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of nodes : ");
        int nodes = sc.nextInt();

      int[] preD = new int[nodes];
      int min = 999, nextNode = 0;
```

```
int[] distance = new int[nodes];
int[][] matrix = new int[nodes][nodes];
int[] visited = new int[nodes];

System.out.println("Enter the cost matrix");

for (int i = 0; i < distance.length; i++)
{
    visited[i] = 0;
    preD[i] = 0;

    for (int j = 0; j < distance.length; j++)
    {
        matrix[i][j] = sc.nextInt();
        if (matrix[i][j]==0)
            matrix[i][j] = 999;
    }
}

distance = matrix[0];
visited[0] = 1;
distance[0] = 0;

for (int counter = 0; counter < nodes; counter++)
{
    min = 999;
    for (int i = 0; i < nodes; i++)
    {
        if (min > distance[i] && visited[i]!=1)
        {
            min = distance[i];
            nextNode = i;
        }
    }
```

```
    visited[nextNode] = 1;
    for (int i = 0; i < nodes; i++)
    {
        if (visited[i]!=1)
        {
            if (min+matrix[nextNode][i] < distance[i])
            {
                distance[i] = min+matrix[nextNode][i];
                preD[i] = nextNode;
            }
        }
    }
}


int j;
for (int i = 0; i < nodes; i++)
{
    if (i!=0)
    {

        System.out.print("Path = " + i);
        j = i;
        do
        {
            j = preD[j];
            System.out.print(" <- " + j);
        }
        while(j != 0);
           System.out.println();
           System.out.print("Cost = " + distance[i]);
    }
    System.out.println("\n");

}
```

```
  }
}
```

OUTPUT :

AIM : Study of Network simulator (NS)

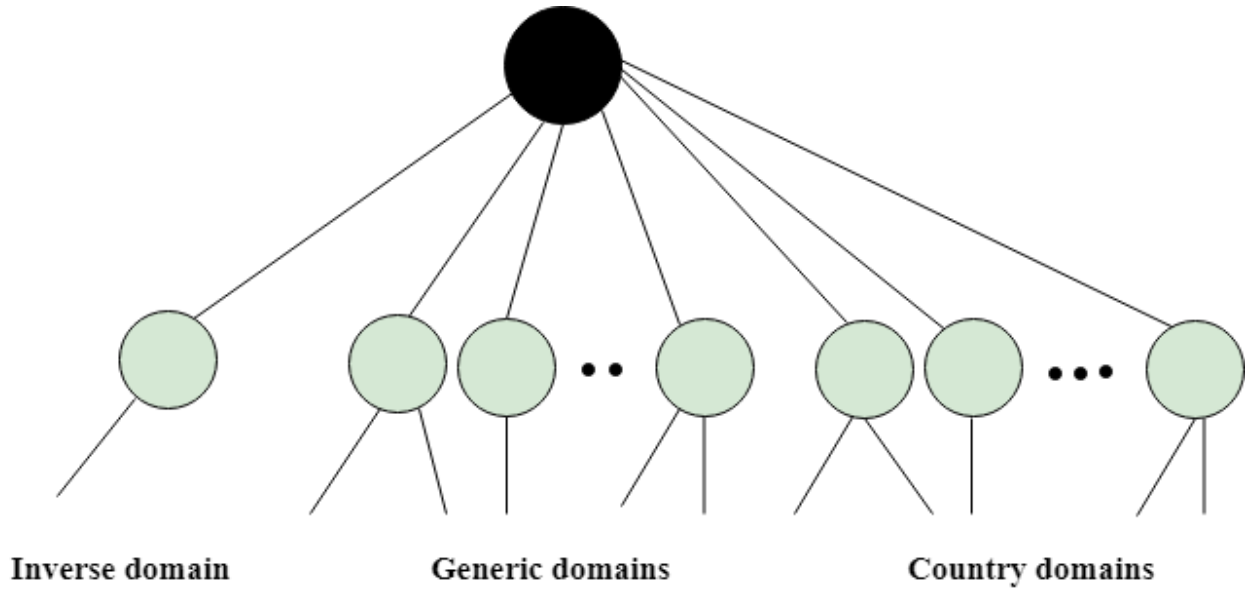
Theory :

DNS

An application layer protocol defines how the application processes running on different systems, pass the messages to each other.

- DNS stands for Domain Name System.
- DNS is a directory service that provides a mapping between the name of a host on the network and its numerical address.
- DNS is required for the functioning of the internet.
- Each node in a tree has a domain name, and a full domain name is a sequence of symbols specified by dots.
- DNS is a service that translates the domain name into IP addresses. This allows the users of networks to utilize user-friendly names when looking for other hosts instead of remembering the IP addresses.
- For example, suppose the FTP site at EduSoft had an IP address of 132.147.165.50, most people would reach this site by specifying ftp.EduSoft.com. Therefore, the domain name is more reliable than IP address.

DNS is a TCP/IP protocol used on different platforms. The domain name space is divided into three different sections: generic domains, country domains, and inverse domain.



Program

```
import java.net.*;
import java.io.*;
import java.util.*;
public class DNS {
public static void main(String[] args) {
 int n;
 BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
 do {
  System.out.println("\n Menu: \n 1. DNS 2. Reverse DNS 3. Exit \n");
  System.out.println("\n Enter your choice");
  n = Integer.parseInt(System.console().readLine());
  if(n==1)
  {
   try
   {
    System.out.println("\n Enter Host Name ");
    String hname=in.readLine();
    InetAddress address;
```

```
    address = InetAddress.getByName(hname);
    System.out.println("Host Name: " + address.getHostName());
    System.out.println("IP: " + address.getHostAddress());
   }
   catch(IOException ioe)
   {
    ioe.printStackTrace();
   }
  }
  if(n==2)
  {
   try
   {
      System.out.println("\n Enter IP address");
      String ipstr = in.readLine();
      InetAddress ia = InetAddress.getByName(ipstr);
      System.out.println("IP: "+ipstr);
      System.out.println("Host Name: " +ia.getHostName());
    }
   catch(IOException ioe)
   {
    ioe.printStackTrace();
   }
  }
 }while(!(n==3));
}
}
```

OUTPUT :