Lab Manual

Computer Networks Lab

Department of Computer Science and Engineering

School of Engineering and Applied Sciences

SRM University AP, Amaravati

Department of Computer Science and Engineering

Vision:

To empower the students to play a vital and leading role as technologist in the rapidly changing arena of Computer Science and Engineering and also prepare them to contribute to the advancement of society and mankind.

Mission:

The Computer Science and Engineering department would impart high quality technical education to help the students to respond to the global happenings and take up research work to create knowledge in the field, through rigorous and comprehensive teaching learning process and inculcate in them the ethical and moral values for the society.

List of Experiments

S. No	Experiments
1	Explain about wire shark and display how to send packets or packets from one layer to another.
2	Implementation of Error Detection Technique using CRC Algorithm.
3	Implementation of Error Correction Technique using Hamming code.
4	Implementation of TCP Client Server programming.
5	Implementation of UDP Client Server Programming.
6	Implementation of Stop and Wait Protocol.
7	Implementation of Sliding Window Protocol.
8	Implementation of Dijkstra Shortest path routing protocol.
9	Implementation of Distance Vector Routing.
10	Implementation of ECHO Command.
11	Java program for PING "Round trip time (RTT)"
12	Implementation of TRACERT Command.
13	Implementation of Subnet Mask.

Experiment-1

Aim: To perform packet analysis and sniffing using Wireshark.

Software used: Wireshark.

Description: Wireshark is a free and open source network protocol analyser that enables users to interactively browse the data traffic on a computer network. Wireshark is a network or protocol analyser (also known as a network sniffer) available for free at the Wireshark website. It is used to analyse the structure of different network protocols and has the ability to demonstrate encapsulation.

Wireshark shares many characteristics with tcp dump. The difference is that it supports a graphical user interface (GUI) and has information filtering features. In addition, Wireshark permits the user to see all the traffic being passed over the network.

Features of Wireshark include:

- Data is analyzed either from the wire over the network connection or from data files that have already captured data packets.
- Supports live data reading and analysis for a wide range of networks (including Ethernet, IEEE 802.11, point-to-point Protocol (PPP) and loopback).
- With the help of GUI or other versions, users can browse captured data networks.
- For programmatically editing and converting the captured files to the edit cap application, users can use command line switches.
- Display filters are used to filter and organize the data display.
- New protocols can be scrutinized by creating plug-ins.
- Captured traffic can also trace Voice over Internet (VoIP) calls over the network.
- When using Linux, it is also possible to capture raw USB traffic.

Observation:

[Protocols in frame: eth: ethertype:ip:**tcp**]

Ethernet II, Src: 04:95: e6:07: fa:28, Dst: d4:25:8b: fb: a3:1f

Type: IPv4 (0x0800)

Internet Protocol Version 4, Src: 202.83.30.234, Dst: 192.168.0.106

Length of data 1452

Transmission Control Protocol (TCP (6)), Src Port: 80, Dst Port: 10015, Seq: 2601985, Ack:192, Len: 1452

TCP segment data (1452 bytes)

Network Header

0000 45 E

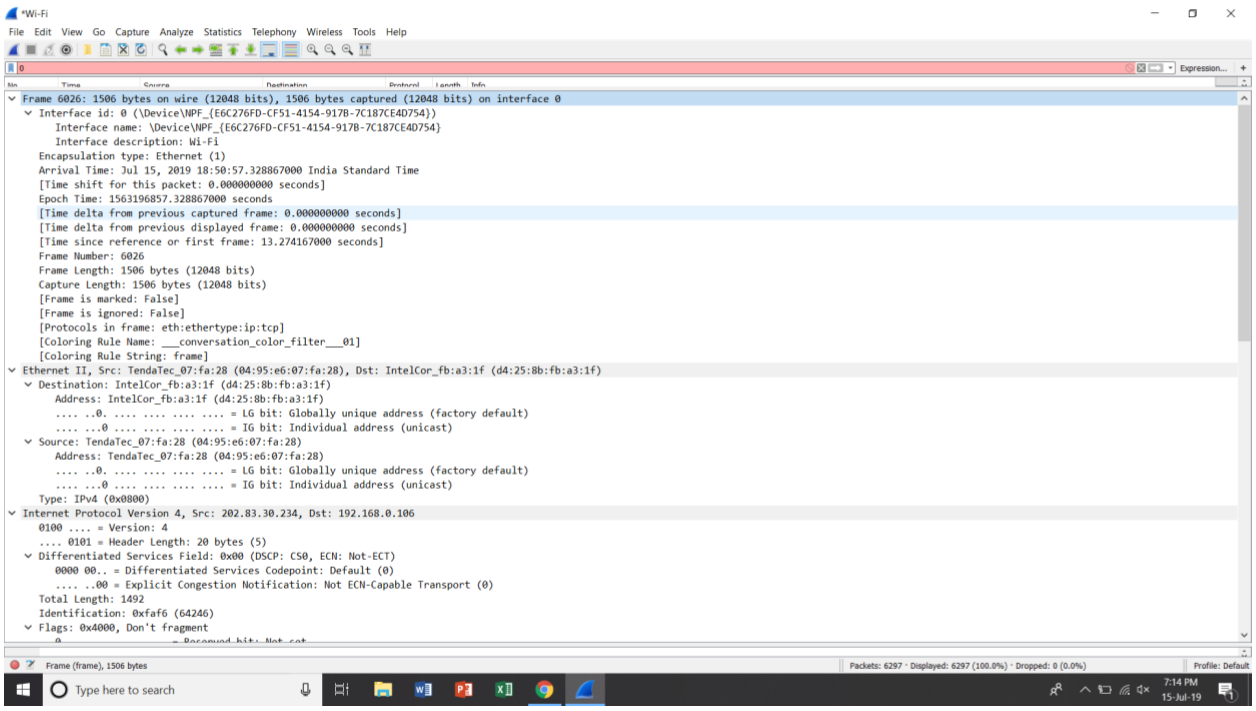
TCP Header

0000 50 P

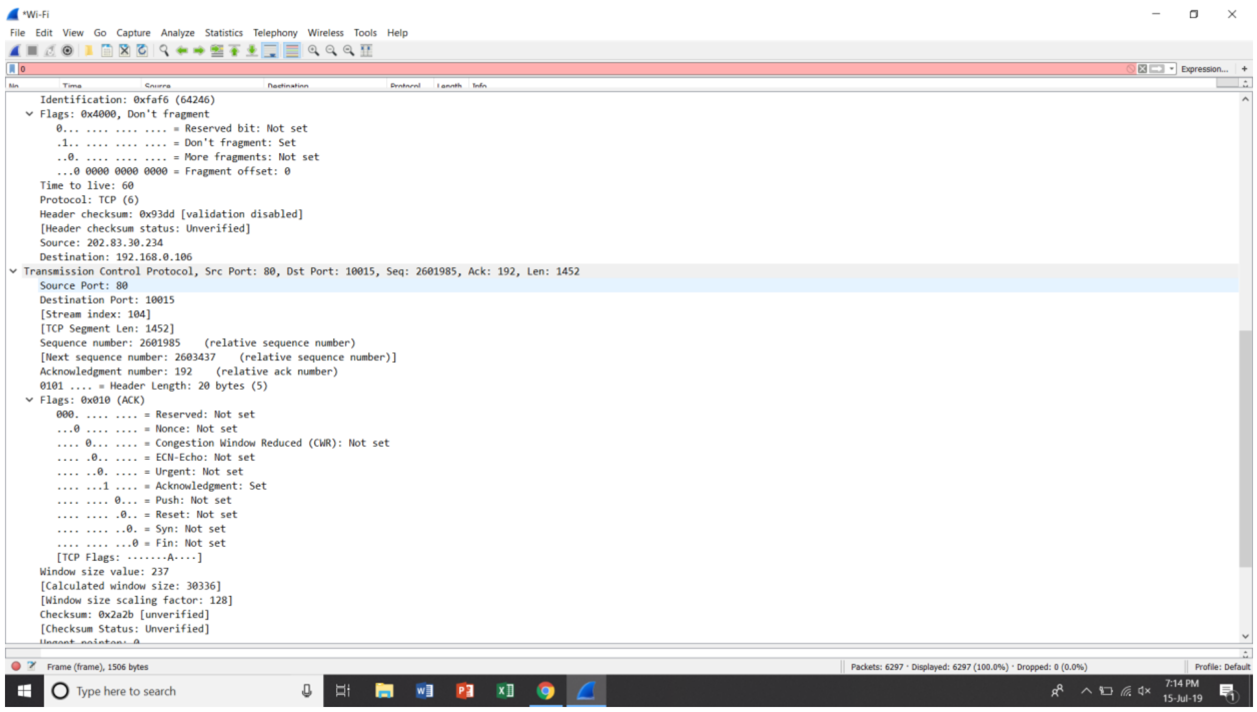
Total Length: 1492

Mac addresses of source and destination(12 bytes), framework type(2 bytes)

Capture Length: 1506 bytes

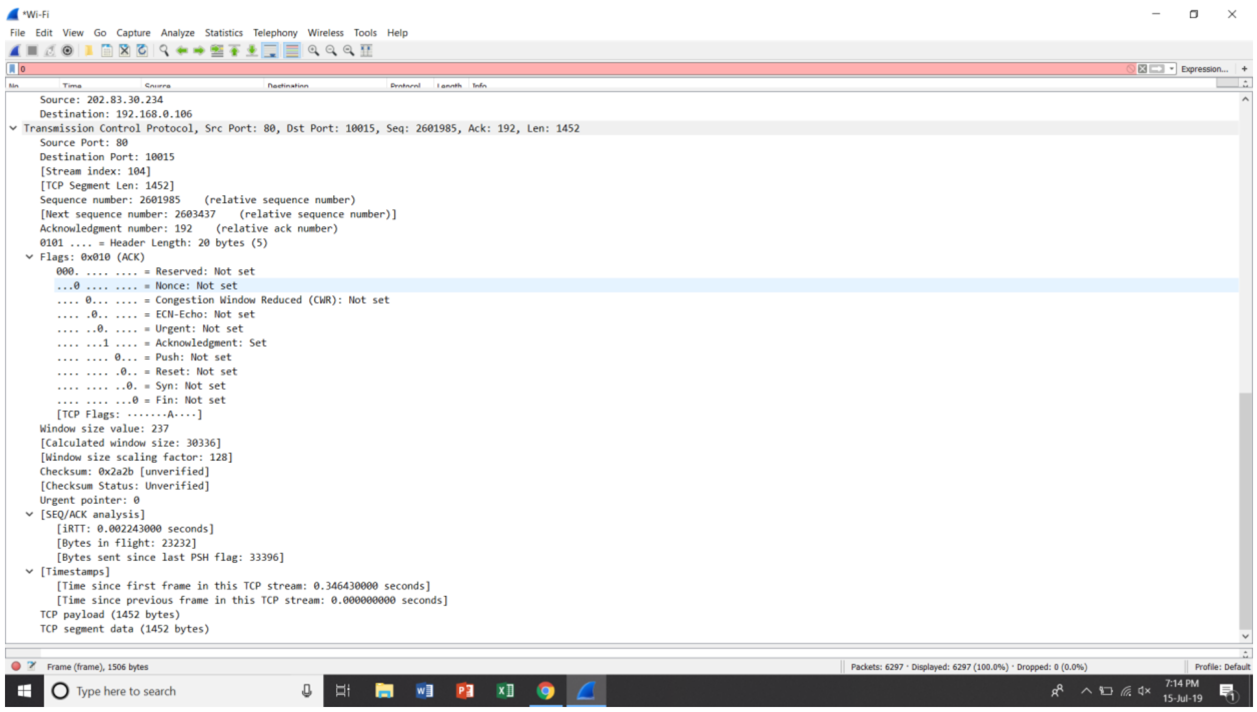

*Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...
Type here to search

```
v Frame 6026: 1506 bytes on wire (12048 bits), 1506 bytes captured (12048 bits) on interface 0
  v Interface id: 0 (\Device\NPF_{E6C276FD-CF51-4154-917B-7C187CE4D754})
      Interface name: \Device\NPF_{E6C276FD-CF51-4154-917B-7C187CE4D754}
      Interface description: Wi-Fi
    Encapsulation type: Ethernet (1)
    Arrival Time: Jul 15, 2019 18:50:57.328867000 India Standard Time
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1563196857.328867000 seconds
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 13.274167000 seconds]
    Frame Number: 6026
    Frame Length: 1506 bytes (12048 bits)
    Capture Length: 1506 bytes (12048 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:tcp]
    [Coloring Rule Name: ___conversation_color_filter___01]
    [Coloring Rule String: frame]
v Ethernet II, Src: TendaTec_07:fa:28 (04:95:e6:07:fa:28), Dst: IntelCor_fb:a3:1f (d4:25:8b:fb:a3:1f)
  v Destination: IntelCor_fb:a3:1f (d4:25:8b:fb:a3:1f)
      Address: IntelCor_fb:a3:1f (d4:25:8b:fb:a3:1f)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  v Source: TendaTec_07:fa:28 (04:95:e6:07:fa:28)
      Address: TendaTec_07:fa:28 (04:95:e6:07:fa:28)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
v Internet Protocol Version 4, Src: 202.83.30.234, Dst: 192.168.0.106
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  v Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
      0000 00.. = Differentiated Services Codepoint: Default (0)
      .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
    Total Length: 1492
    Identification: 0xfaf6 (64246)
  v Flags: 0x4000, Don't fragment

Frame (frame), 1506 bytes          Packets: 6297 · Displayed: 6297 (100.0%) · Dropped: 0 (0.0%)          Profile: Default
7:14 PM
15-Jul-19
```


*Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...
Type here to search
7:14 PM
15-Jul-19

```
    Identification: 0xfaf6 (64246)
  v Flags: 0x4000, Don't fragment
        0... .... .... .... = Reserved bit: Not set
        .1.. .... .... .... = Don't fragment: Set
        ..0. .... .... .... = More fragments: Not set
        ...0 0000 0000 0000 = Fragment offset: 0
    Time to live: 60
    Protocol: TCP (6)
    Header checksum: 0x93dd [validation disabled]
    [Header checksum status: Unverified]
    Source: 202.83.30.234
    Destination: 192.168.0.106
v Transmission Control Protocol, Src Port: 80, Dst Port: 10015, Seq: 2601985, Ack: 192, Len: 1452
    Source Port: 80
    Destination Port: 10015
    [Stream index: 104]
    [TCP Segment Len: 1452]
    Sequence number: 2601985    (relative sequence number)
    [Next sequence number: 2603437    (relative sequence number)]
    Acknowledgment number: 192    (relative ack number)
    0101 .... = Header Length: 20 bytes (5)
  v Flags: 0x010 (ACK)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... .... 0... = Push: Not set
        .... .... .0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
        .... .... ...0 = Fin: Not set
        [TCP Flags: ·······A····]
    Window size value: 237
    [Calculated window size: 30336]
    [Window size scaling factor: 128]
    Checksum: 0x2a2b [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
```

```
Frame (frame), 1506 bytes    Packets: 6297 · Displayed: 6297 (100.0%) · Dropped: 0 (0.0%)    Profile: Default
```


*Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...
Type here to search
7:14 PM
15-Jul-19

```
    Source: 202.83.30.234
    Destination: 192.168.0.106
v Transmission Control Protocol, Src Port: 80, Dst Port: 10015, Seq: 2601985, Ack: 192, Len: 1452
    Source Port: 80
    Destination Port: 10015
    [Stream index: 104]
    [TCP Segment Len: 1452]
    Sequence number: 2601985    (relative sequence number)
    [Next sequence number: 2603437    (relative sequence number)]
    Acknowledgment number: 192    (relative ack number)
    0101 .... = Header Length: 20 bytes (5)
  v Flags: 0x010 (ACK)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... .... 0... = Push: Not set
        .... .... .0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
        .... .... ...0 = Fin: Not set
        [TCP Flags: ·······A····]
    Window size value: 237
    [Calculated window size: 30336]
    [Window size scaling factor: 128]
    Checksum: 0x2a2b [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  v [SEQ/ACK analysis]
        [iRTT: 0.002243000 seconds]
        [Bytes in flight: 23232]
        [Bytes sent since last PSH flag: 33396]
  v [Timestamps]
        [Time since first frame in this TCP stream: 0.346430000 seconds]
        [Time since previous frame in this TCP stream: 0.000000000 seconds]
    TCP payload (1452 bytes)
    TCP segment data (1452 bytes)
```

```
Frame (frame), 1506 bytes    Packets: 6297 · Displayed: 6297 (100.0%) · Dropped: 0 (0.0%)    Profile: Default
```

Experiment-2

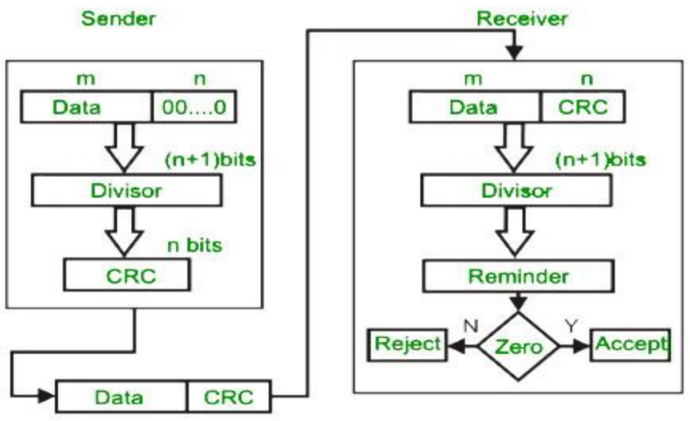
Aim: Implementation of Error Detection Technique using CRC Algorithm.

Description: A **cyclic redundancy check** (**CRC**) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems get a short check value attached, based on the remainder of a polynomial division of their contents. On retrieval, the calculation is repeated and, in the event the check values do not match, corrective action can be taken against data corruption.

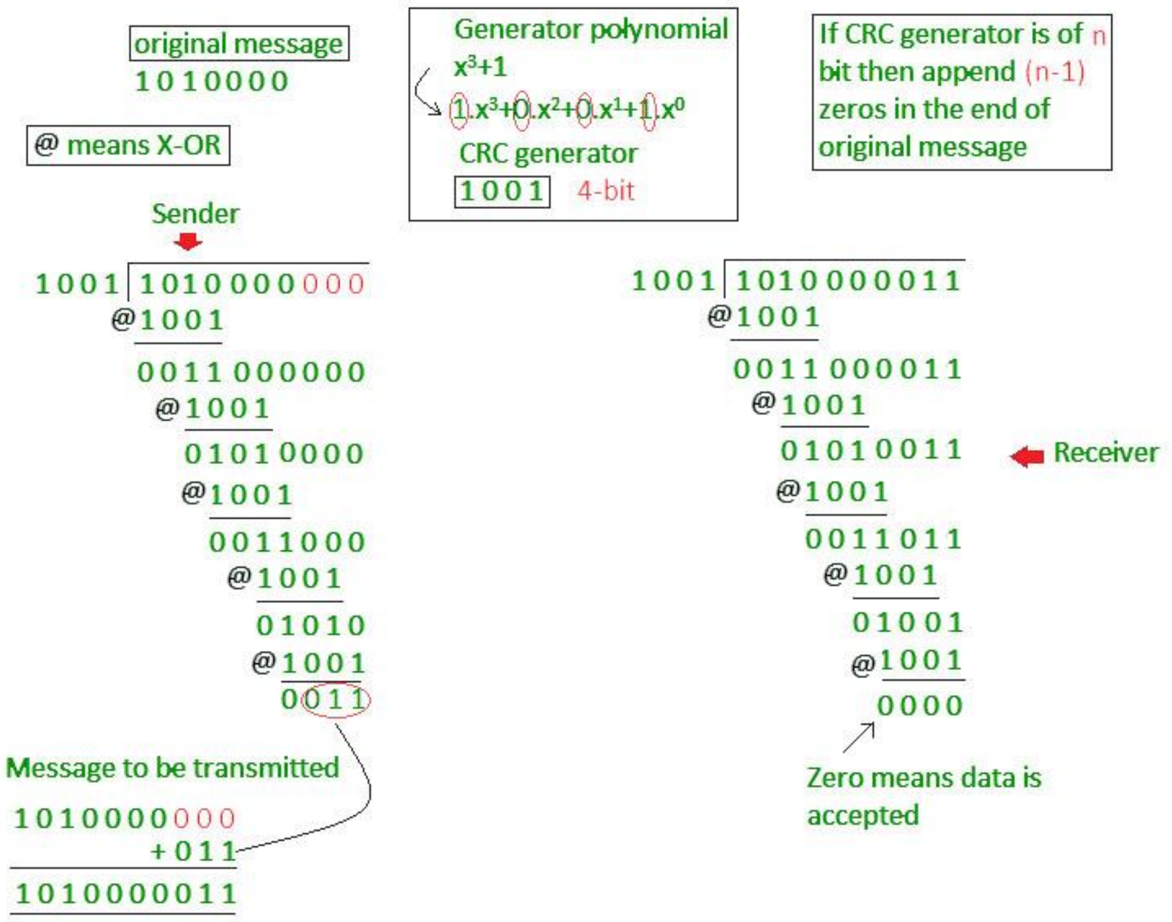
Whenever a message is transmitted, it may get scrambled by noise or data may get corrupted. To avoid this, we use error-detecting codes which are additional data added to a given digital message to help us detect if any error has occurred during transmission of the message.

Cyclic redundancy check (CRC)

- Unlike checksum scheme, which is based on addition, CRC is based on binary division.
- In CRC, a sequence of redundant bits, called cyclic redundancy check bits, are appended to the end of data unit so that the resulting data unit becomes exactly divisible by a second, predetermined binary number.
- At the destination, the incoming data unit is divided by the same number. If at this step there is no remainder, the data unit is assumed to be correct and is therefore accepted.
- A remainder indicates that the data unit has been damaged in transit and therefore must be rejected.



Example:


original message
1010000
@ means X-OR
Generator polynomial
x³+1
1.x³+0.x²+0.x¹+1.x⁰
CRC generator
1001 4-bit
If CRC generator is of n bit then append (n-1) zeros in the end of original message
Sender
1001 | 1010000000
@1001
0011000000
@1001
01010000
@1001
0011000
@1001
01010
@1001
0011
Message to be transmitted
1010000000
+011
1010000011
1001 | 1010000011
@1001
0011000011
@1001
01010011
@1001
0011011
@1001
01001
@1001
0000
Receiver
Zero means data is accepted

Program:

```
import java.util.*;
class CRC
    {
        public static void main(String args[])
    {
                Scanner scan = new Scanner(System.in);
                int n;
                System.out.println("Enter the size of the data:");
                n = scan.nextInt();
                int data[] = new int[n];
                System.out.println("Enter the data, bit by bit:");
                for(int i=0 ; i < n ; i++)
            {
                        System.out.println("Enter bit number " + (n-i) + ":");
                        data[i] = scan.nextInt();
                }
                System.out.println("Enter the size of the divisor:");
                n = scan.nextInt();
                int divisor[] = new int[n];
                System.out.println("Enter the divisor, bit by bit:");
                for(int i=0 ; i < n ; i++)
            {
                        System.out.println("Enter bit number " + (n-i) + ":");
                        divisor[i] = scan.nextInt();
                }
                int remainder[] = divide(data, divisor);
                for(int i=0 ; i < remainder.length-1 ; i++)
            {
                        System.out.print(remainder[i]);
                }
            System.out.println("\nThe CRC code generated is:");
            for(int i=0 ; i < data.length ; i++)
        {
                System.out.print(data[i]);
            }
            for(int i=0 ; i < remainder.length-1 ; i++)
        {
                System.out.print(remainder[i]);
            }
            System.out.println();
            int sent_data[] = new int[data.length + remainder.length - 1];
            System.out.println("Enter the data to be sent:");
            for(int i=0 ; i < sent_data.length ; i++)
        {
                System.out.println("Enter bit number " + (sent_data.length-i)+ ":");
                sent_data[i] = scan.nextInt();
            }
            receive(sent_data, divisor);
    }
    static int[] divide(int old_data[], int divisor[])
  {
            int remainder[] , i;
            int data[] = new int[old_data.length + divisor.length];
            System.arraycopy(old_data, 0, data, 0, old_data.length);
            remainder = new int[divisor.length];
            System.arraycopy(data, 0, remainder, 0, divisor.length);
            for(i=0 ; i < old_data.length ; i++)
        {
                System.out.println((i+1) + ".) First data bit is : "+ remainder[0]);
                System.out.print("Remainder : ");
                if(remainder[0] == 1)
            {
                        for(int j=1 ; j < divisor.length ; j++)
              {
                             remainder[j-1] = exor(remainder[j], divisor[j]);
                             System.out.print(remainder[j-1]);
                        }
                 }
                 else {
                        for(int j=1 ; j < divisor.length ; j++)
              {
                             remainder[j-1] = exor(remainder[j], 0);
                             System.out.print(remainder[j-1]);
                        }
                 }
                 remainder[divisor.length-1] = data[i+divisor.length];
                 System.out.println(remainder[divisor.length-1]);
            }
            return remainder;
       }

       static int exor(int a, int b)
     {
            if(a == b)
         {
                 return 0;
            }
            return 1;
       }

       static void receive(int data[], int divisor[])
     {
```

```
			int remainder[] = divide(data, divisor);
			for(int i=0 ; i < remainder.length ; i++)
		{
				if(remainder[i] != 0)
			{
					System.out.println("There is an error in received data...");
					return;
				}
			}
			System.out.println("Data was received without any error.");
	}
}
```

Output:

```
C:\Users\LAB602_44\Desktop>javac CRC.java
C:\Users\LAB602_44\Desktop>java CRC
Enter the size of the data:
5
Enter the data, bit by bit:
Enter bit number 5:
1
Enter bit number 4:
0
Enter bit number 3:
1
Enter bit number 2:
1
Enter bit number 1:
0
Enter the size of the divisor:
3
Enter the divisor, bit by bit:
Enter bit number 3:
```

```
1
Enter bit number 2:
1
Enter bit number 1:
0
1.) First data bit is: 1
Remainder: 111
2.) First data bit is: 1
Remainder: 010
3.) First data bit is: 0
Remainder: 100
4.) First data bit is: 1
Remainder: 100
5.) First data bit is: 1
Remainder: 100
10
The CRC code generated is:
1011010
Enter the data to be sent:
Enter bit number 7:
1
Enter bit number 6:
0
Enter bit number 5:
1
Enter bit number 4:
1
Enter bit number 3:
0
Enter bit number 2:
1
Enter bit number 1:
0
1.) First data bit is: 1
```

```
Remainder: 111
2.) First data bit is: 1
Remainder: 010
3.) First data bit is: 0
Remainder: 101
4.) First data bit is: 1
Remainder: 110
5.) First data bit is: 1
Remainder: 000
6.) First data bit is: 0
Remainder: 000
7.) First data bit is: 0
Remainder: 000
Data was received without any error.
```

Experiment-3

Aim: Implementation of Error Correction Technique using Hamming code.

Description: Hamming code is a set of error-correction codes that can be used to **detect and correct the errors** that can occur when the data is moved or stored from the sender to the receiver. It is **technique developed by R.W. Hamming for error correction**.

Redundant bits –

Redundant bits are extra binary bits that are generated and added to the information-carrying bits of data transfer to ensure that no bits were lost during the data transfer. The number of redundant bits can be calculated using the following formula:

$2^r \geq m + r + 1$

where, r = redundant bit, m = data bit

Suppose the number of data bits is 7, then the number of redundant bits can be calculated using: $=2^4 \geq 7+4+1$Thus, the number of redundant bits= 4

Parity-bits:

A parity bit is a bit appended to a data of binary bits to ensure that the total number of 1's in the data are even or odd. Parity bits are used for error detection. There are two types of parity bits:

Even-parity-bit:

In the case of even parity, for a given set of bits, the number of 1's are counted. If that count is odd, the parity bit value is set to 1, making the total count of occurrences of 1's an even number. If the total number of 1's in a given set of bits is already even, the parity bit's value is 0.

Odd-Parity-bit:

In the case of odd parity, for a given set of bits, the number of 1's are counted. If that count is even, the parity bit value is set to 1, making the total count of occurrences of 1's an odd number. If the total number of 1's in a given set of bits is already odd, the parity bit's value is 0.

General Algorithm for Hamming Code:

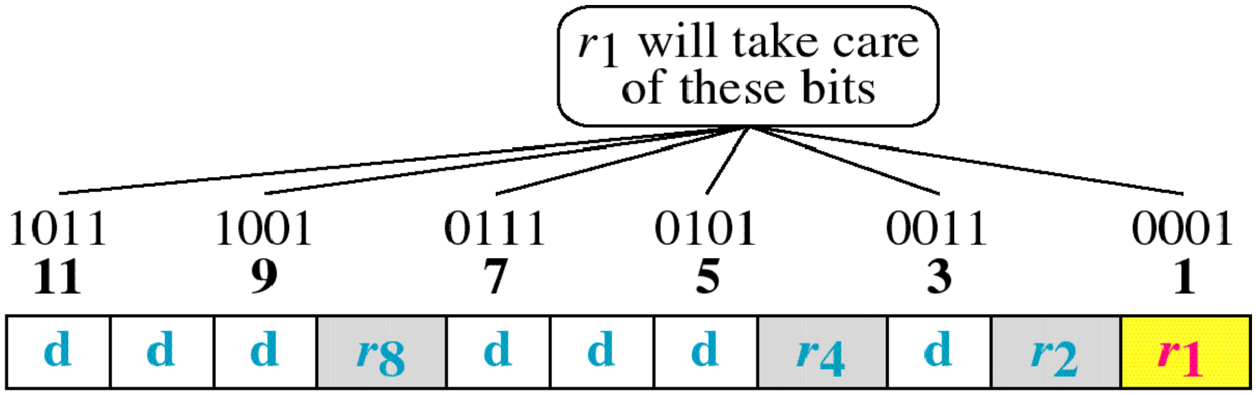
The Hamming Code is simply the use of extra parity bits to allow the identification of an error.

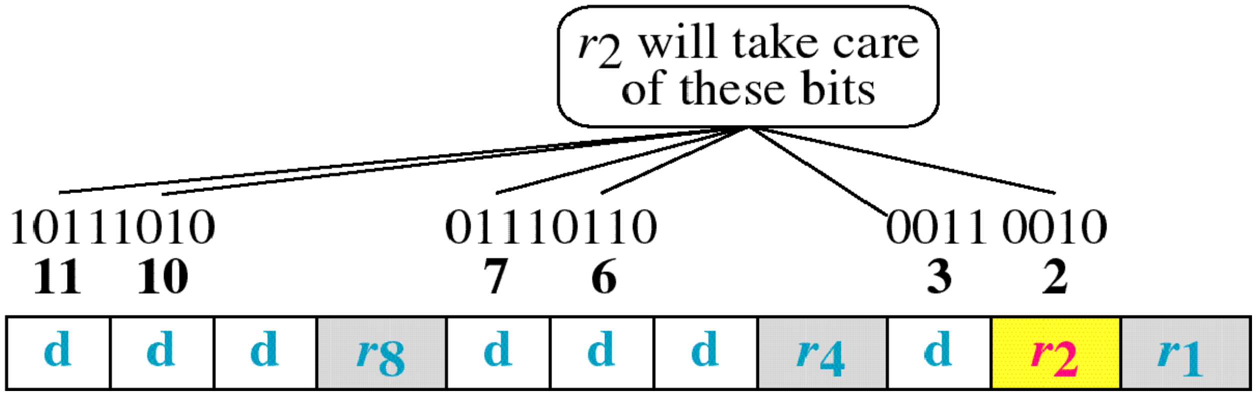
1. Write the bit positions starting from 1 in binary form (1, 10, 11, 100, etc).
2. All the bit positions that are a power of 2 are marked as parity bits (1, 2, 4, 8, etc).
3. All the other bit positions are marked as data bits.
4. Each data bit is included in a unique set of parity bits, as determined its bit position in binary form.

   a. Parity bit 1 covers all the bits positions whose binary representation includes a 1 in the least significant position (1, 3, 5, 7, 9, 11, etc).
   b. Parity bit 2 covers all the bits positions whose binary representation includes a 1 in the second position from the least significant bit (2, 3, 6, 7, 10, 11, etc).
   c. Parity bit 4 covers all the bits positions whose binary representation includes a 1 in the third position from the least significant bit (4–7, 12–15, 20–23, etc).
   d. Parity bit 8 covers all the bits positions whose binary representation includes a 1 in the fourth position from the least significant bit bits (8–15, 24–31, 40–47, etc).
   e. In general, each parity bit covers all bits where the bitwise AND of the parity position and the bit position is, non-zero.
5. Since we check for even parity set a parity bit to 1 if the total number of ones in the positions it checks is odd.
6. Set a parity bit to 0 if the total number of ones in the positions it checks is even.

Hamming Code Example:

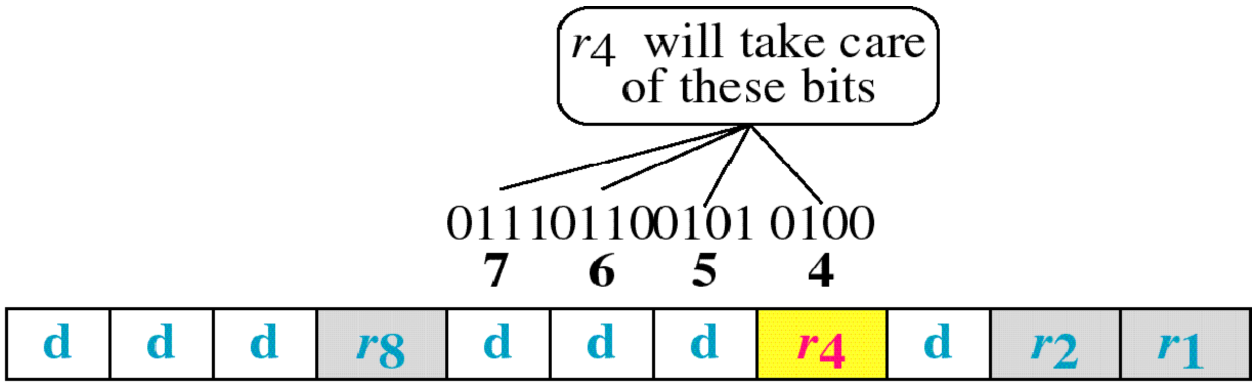
r1 is a parity check on every data bit whose position is xxx1

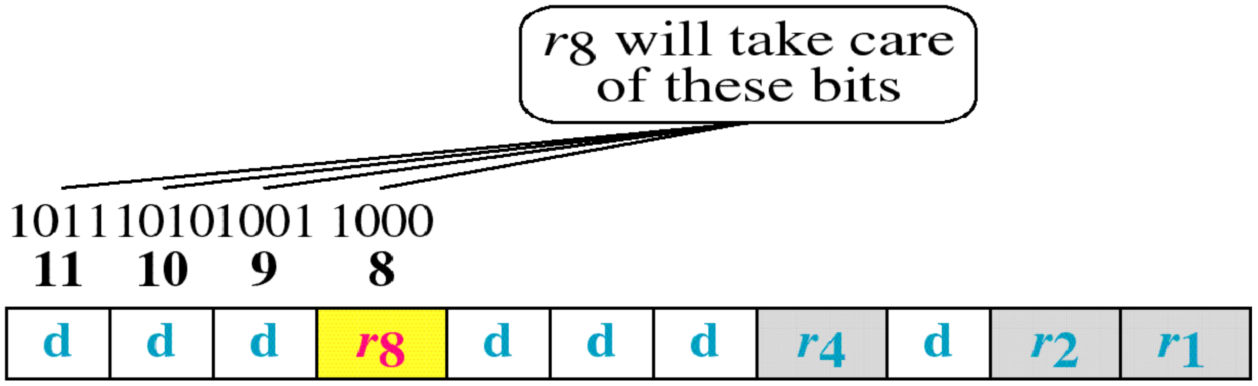




r2 is a parity check on every data bit whose position is xx1x

r4 is a parity check on every data bit whose position is x1xx





r8 is a parity check on every data bit whose position is 1xxx

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

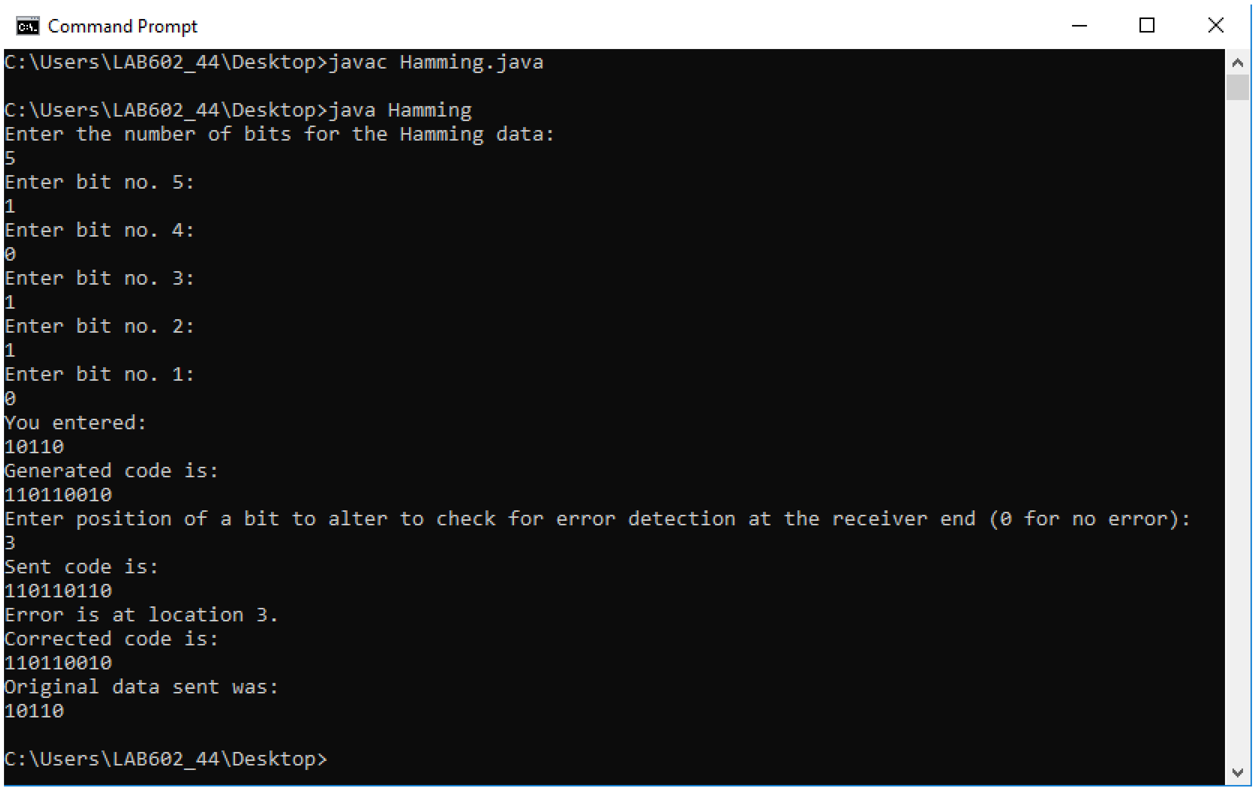
```
                int error = scan.nextInt();
                if(error != 0)
        {
                        b[error-1] = (b[error-1]+1)%2;
                }
                System.out.println("Sent code is:");
                for(int i=0 ; i < b.length ; i++)
        {
                        System.out.print(b[b.length-i-1]);
                }
                System.out.println();
                receive(b, b.length - a.length);
        }
        static int[] generateCode(int a[])
    {
                int b[];
                int i=0, parity_count=0 ,j=0, k=0;
                while(i < a.length)
        {
                        if(Math.pow(2,parity_count) == i+parity_count + 1)
            {
                                parity_count++;
                        }
                        else
            {
                                i++;
                        }
                }
                b = new int[a.length + parity_count];
                for(i=1 ; i <= b.length ; i++)
        {
```

```
                        if(Math.pow(2, j) == i)
                {
                                b[i-1] = 2;
                                j++;
                        }
                        else
                {
                                b[k+j] = a[k++];
                        }
                  }
                  for(i=0 ; i < parity_count ; i++)
             {
                        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)
                {
                                int k = i+1;
                                String s = Integer.toBinaryString(k);
                                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)
     {
               int power;
               int parity[] = new int[parity_count];
               String syndrome = new String();
               for(power=0 ; power < parity_count ; power++)
          {
                    for(int i=0 ; i < a.length ; i++)
               {
                         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;
```

```
            }
            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.");
            }
            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:

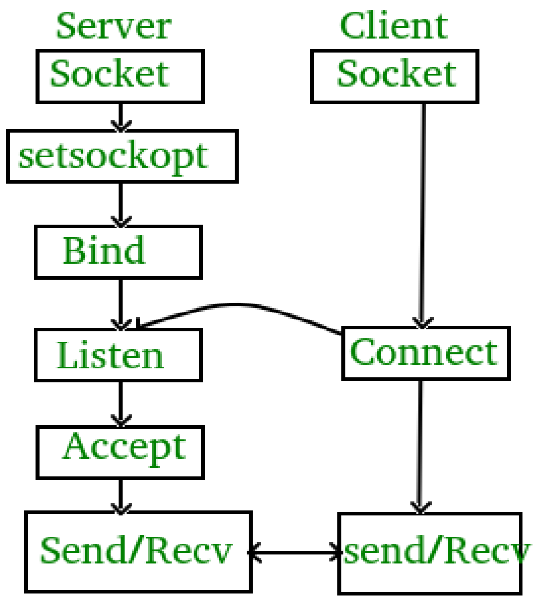


```
C:\Users\LAB602_44\Desktop>javac Hamming.java

C:\Users\LAB602_44\Desktop>java Hamming
Enter the number of bits for the Hamming data:
5
Enter bit no. 5:
1
Enter bit no. 4:
0
Enter bit no. 3:
1
Enter bit no. 2:
1
Enter bit no. 1:
0
You entered:
10110
Generated code is:
110110010
Enter position of a bit to alter to check for error detection at the receiver end (0 for no error):
3
Sent code is:
110110110
Error is at location 3.
Corrected code is:
110110010
Original data sent was:
10110

C:\Users\LAB602_44\Desktop>
```

Experiment-4

Aim: Implementation of TCP Client Server programming.

Description: TCP is suited for applications that require high reliability, and transmission time is relatively less critical. It is used by other protocols like HTTP, HTTPs, FTP, SMTP, Telnet. TCP rearranges data packets in the order specified. There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent. TCP does Flow Control and requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control. It also does error checking and error recovery. Erroneous packets are retransmitted from the source to the destination.



TCP Server –

1. using create(), Create TCP socket.
2. using bind(), Bind the socket to server address.
3. using listen(), put the server socket in a passive mode, where it waits for the client to approach the server to make a connection
4. using accept(), At this point, connection is established between client and server, and they are ready to transfer data.
5. Go back to Step 3.

TCP Client –

1. Create TCP socket.
2. connect newly created client socket to server.

Program for TCPServer:

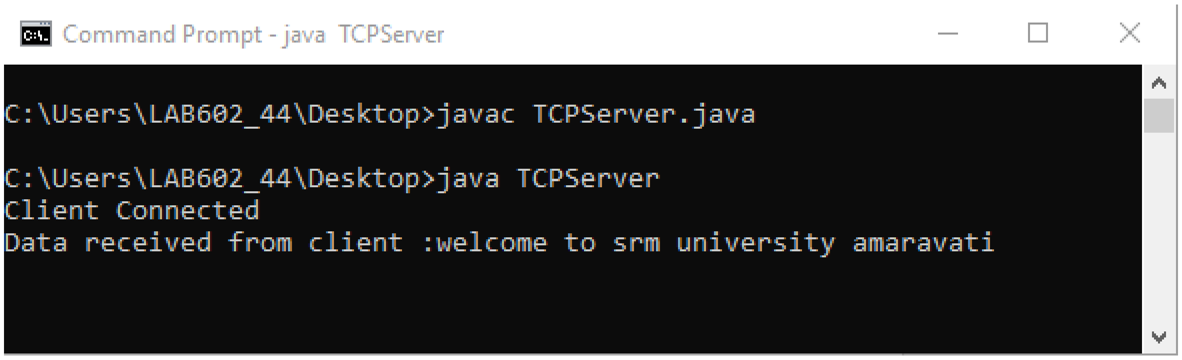
```
import java.io.*;
import java.net.*;
class TCPServer
{
public static void main(String argv[]) throws Exception
{
String clientSentence;
String capitalizedSentence;
ServerSocket welcomeSocket=new ServerSocket(6789);
while(true){
Socket connectionSocket=welcomeSocket.accept();
System.out.println("Client Connected");
BufferedReader inFromClient=new BufferedReader(new
InputStreamReader(connectionSocket.getInputStream()));
DataOutputStream outToClient=new DataOutputStream(connectionSocket.getOutputStream());
clientSentence=inFromClient.readLine();
System.out.println("Data received from client :" + clientSentence);
capitalizedSentence=clientSentence.toUpperCase()+'\n';
outToClient.writeBytes(capitalizedSentence);
}
}
}
```

Program for TCPClient:

```
import java.io.*;
import java.net.*;
class TCPClient {
public static void main(String argv[]) throws Exception
{
String sentence;
String modifiedSentence;
System.out.println("Enter lower case sentence from keyboard \n");
BufferedReader inFromUser=new BufferedReader(new InputStreamReader(System.in));
Socket clientSocket=new Socket("localHost", 6789);
DataOutputStream outToServer=new DataOutputStream(clientSocket.getOutputStream());
BufferedReader inFromServer=new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
sentence=inFromUser.readLine();
outToServer.writeBytes(sentence + '\n');
modifiedSentence = inFromServer.readLine();
System.out.println("FROM SERVER: " + modifiedSentence);
clientSocket.close();
}
}
```

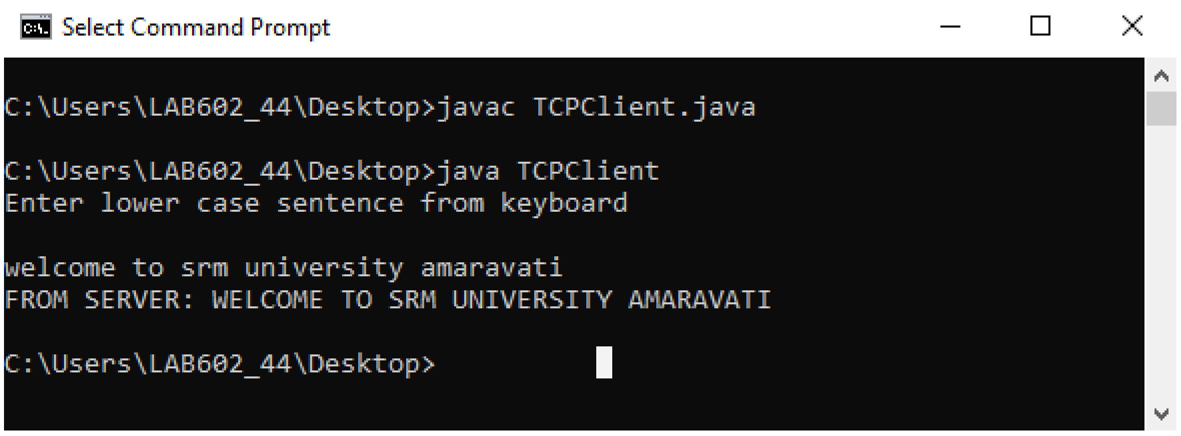
Output:

TCPserver:



```
C:\Users\LAB602_44\Desktop>javac TCPServer.java

C:\Users\LAB602_44\Desktop>java TCPServer
Client Connected
Data received from client :welcome to srm university amaravati
```

TCPClinet:

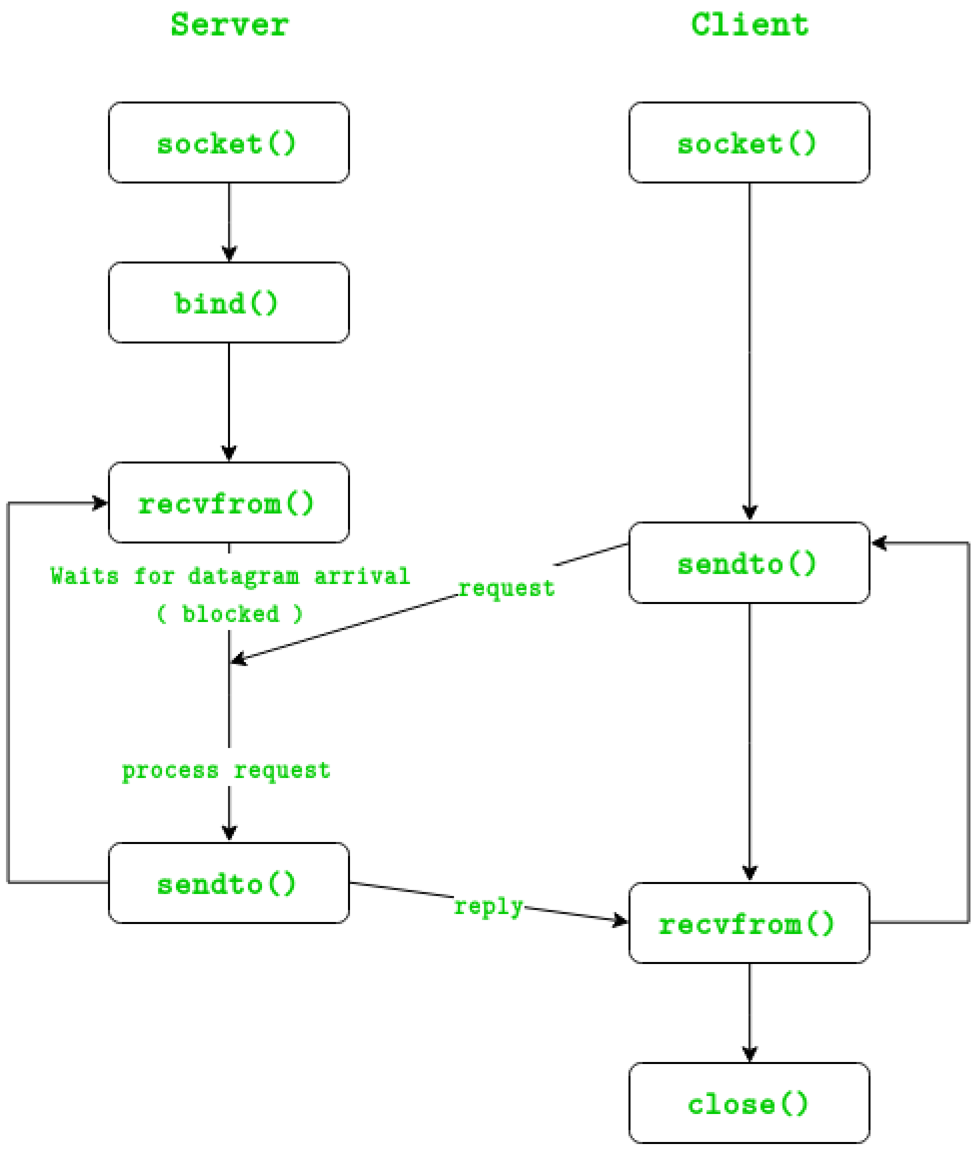


```
C:\Users\LAB602_44\Desktop>javac TCPClient.java

C:\Users\LAB602_44\Desktop>java TCPClient
Enter lower case sentence from keyboard

welcome to srm university amaravati
FROM SERVER: WELCOME TO SRM UNIVERSITY AMARAVATI

C:\Users\LAB602_44\Desktop>
```

Experiment-5

Aim: Implementation of UDP Client Server programming.

Description: In UDP, the client does not form a connection with the server like in TCP and instead just sends a datagram. Similarly, the server need not accept a connection and just waits for datagrams to arrive. Datagrams upon arrival contain the address of sender which the server uses to send data to the correct client.



UDP Server:

1. Create UDP socket.
2. Bind the socket to server address.
3. Wait until datagram packet arrives from client.
4. Process the datagram packet and send a reply to client.
5. Go back to Step 3.

UDP Client:

1. Create UDP socket.
2. Send message to server.
3. Wait until response from server is received.
4. Process reply and go back to step 2, if necessary.
5. Close socket descriptor and exit.

Program for UDPServer:

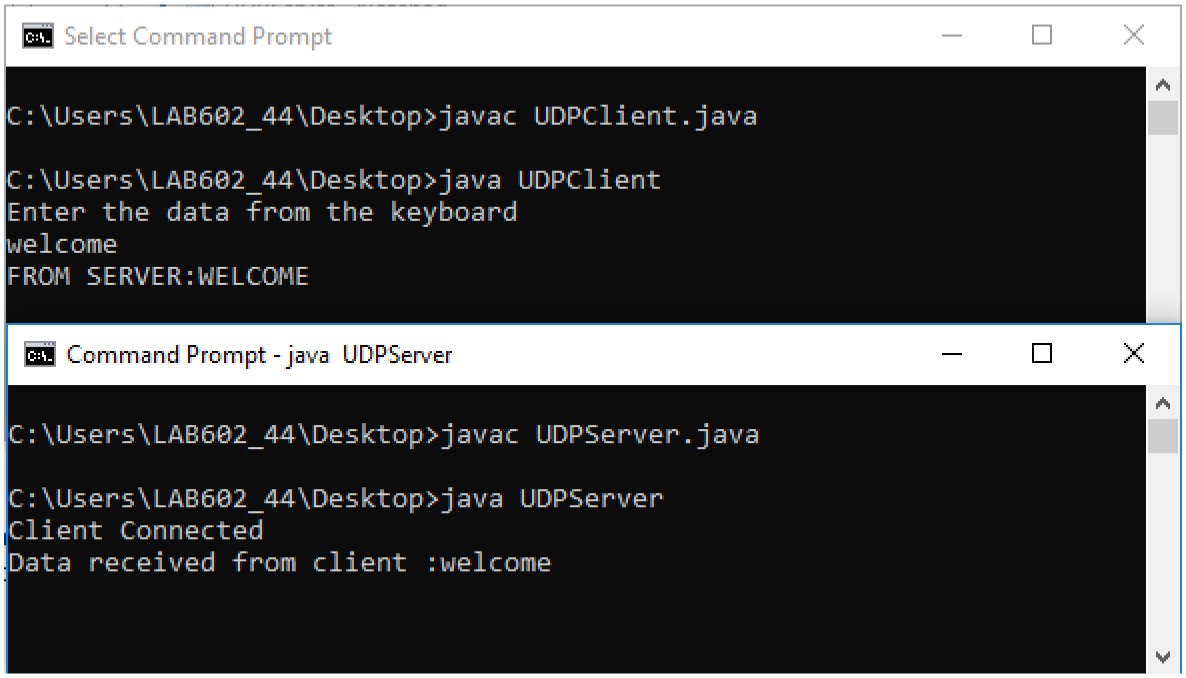
```
import java.io.*;
import java.net.*;
class UDPServer{
public static void main(String args[]) throws Exception
{
DatagramSocket serverSocket=new DatagramSocket(9876);
byte[] receiveData = new byte[1024];
byte[] sendData  = new byte[1024];
while(true){
DatagramPacket receivePacket=new DatagramPacket(receiveData, receiveData.length);
serverSocket.receive(receivePacket);
System.out.println("Client Connected");
String sentence = new String(receivePacket.getData());
InetAddress IPAddress = receivePacket.getAddress();
int port = receivePacket.getPort();
System.out.println("Data received from client :" +sentence);
String capitalizedSentence = sentence.toUpperCase();
sendData = capitalizedSentence.getBytes();
DatagramPacket sendPacket=new DatagramPacket(sendData, sendData.length, IPAddress, port);
serverSocket.send(sendPacket);
}
}
}
```

Program for UDPClient:

```
import java.io.*;
import java.net.*;
class UDPClient{
public static void main(String args[]) throws Exception
{
System.out.println("Enter the data from the keyboard");
BufferedReader inFromUser=new BufferedReader(new InputStreamReader(System.in));
DatagramSocket clientSocket=new DatagramSocket();
InetAddress IPAddress=InetAddress.getByName("localHost");
byte[] sendData = new byte[1024];
byte[] receiveData = new byte[1024];
String sentence = inFromUser.readLine();
sendData = sentence.getBytes();
DatagramPacket sendPacket=new DatagramPacket(sendData, sendData.length, IPAddress, 9876);
clientSocket.send(sendPacket);
DatagramPacket receivePacket=new DatagramPacket(receiveData, receiveData.length);
clientSocket.receive(receivePacket);
String modifiedSentence=new String(receivePacket.getData());
System.out.println("FROM SERVER:" + modifiedSentence);
clientSocket.close();
```

} }

Output:



```
C:\Users\LAB602_44\Desktop>javac UDPClient.java

C:\Users\LAB602_44\Desktop>java UDPClient
Enter the data from the keyboard
welcome
FROM SERVER:WELCOME
```

```
C:\Users\LAB602_44\Desktop>javac UDPServer.java

C:\Users\LAB602_44\Desktop>java UDPServer
Client Connected
Data received from client :welcome
```

Experiment-6

Aim: Implementation of Stop and Wait Protocol.

Description: In this method of flow control, the sender sends a single frame to receiver & waits for an acknowledgment.

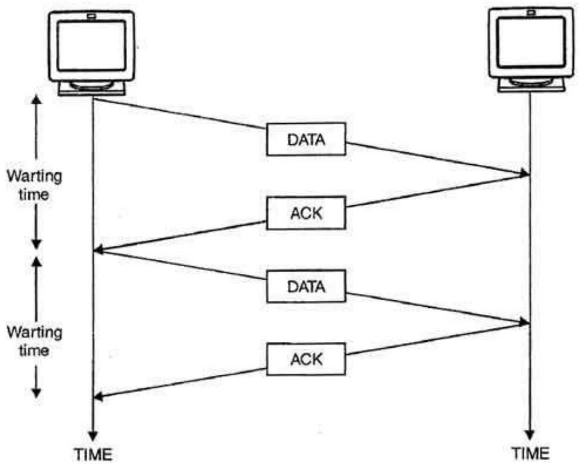
• The next frame is sent by sender only when acknowledgment of previous frame is received.

• This process of sending a frame & waiting for an acknowledgment continues as long as the sender has data to send.

• To end up the transmission sender transmits end of transmission (EOT) frame.

• The main advantage of stop & wait protocols is its accuracy. Next frame is transmitted only when the first frame is acknowledged. So, there is no chance of frame being lost.

• The main disadvantage of this method is that it is inefficient. It makes the transmission process slow. In this method single frame travels from source to destination and single acknowledgment travels from destination to source. As a result, each frame sent and received uses the entire time needed to traverse the link. Moreover, if two devices are distance apart, a lot of time is wasted waiting for ACKs that leads to increase in total transmission time.



Stop & Wait Method.

Program for BitServer:

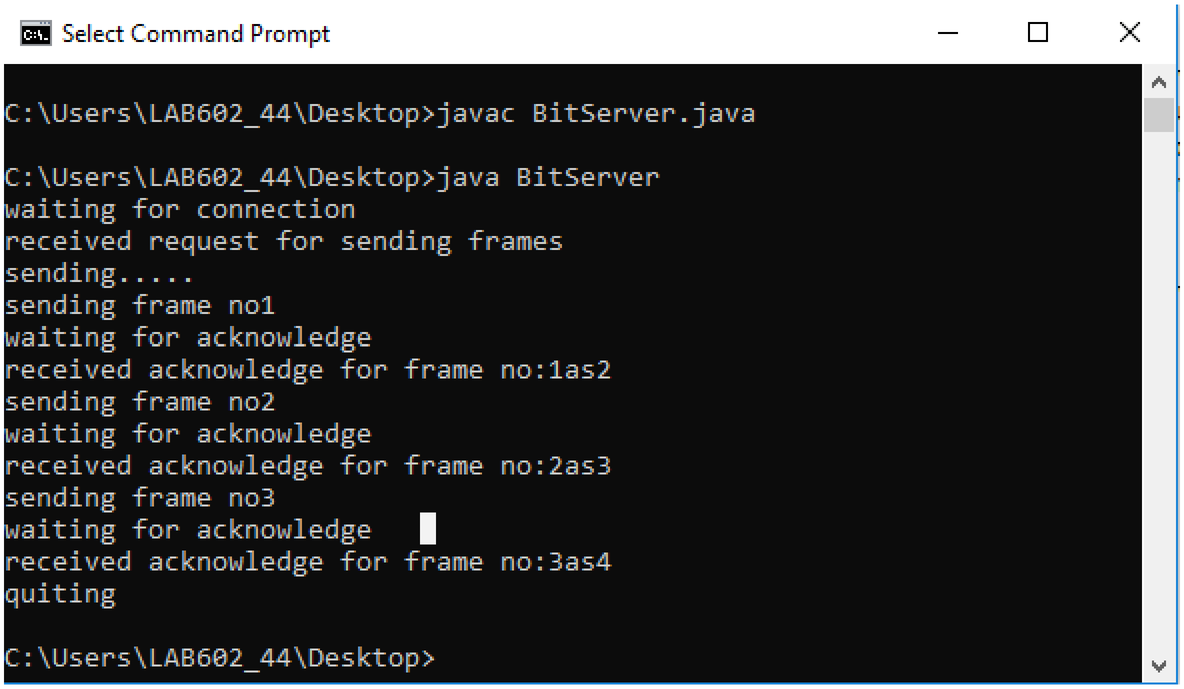
```
import java.lang.System;
import java.net.*;
import java.io.*;
class bitserver
{
public static void main(String[] args)
{
try
{
BufferedInputStream in;
ServerSocket Serversocket=new ServerSocket(500);
System.out.println("waiting for connection");
Socket client=Serversocket.accept();
System.out.println("received request for sending frames");
in=new BufferedInputStream(client.getInputStream());
DataOutputStream out=new DataOutputStream(client.getOutputStream());
int p=in.read();
System.out.println("sending.....");
for(int i=1;i<=p;++i)
{
System.out.println("sending frame no"+i);
out.write(i);
out.flush();
System.out.println("waiting for acknowledge");
Thread.sleep(5000);
int a=in.read();
System.out.println("received acknowledge for frame no:"+i+ "as" +a);
}
  out.flush();
in.close();
out.close();
client.close();
Serversocket.close();
System.out.println("quiting");
}
catch(IOException e)
{
System.out.println(e);
}
catch(InterruptedException e)
{
System.out.println(“Error occured”);
}
}
}
```

Program for Client:

```
import java.lang.System;
import java.net.*;
import java.io.*;
import java.math.*;
class client
{
public static void main(String a[])
{
try
{
InetAddress addr=InetAddress.getByName("Localhost");
System.out.println(addr);
Socket connection=new Socket(addr,500);
DataOutputStream out=new DataOutputStream(connection.getOutputStream());
BufferedInputStream in=new BufferedInputStream(connection.getInputStream());
BufferedInputStream inn=new BufferedInputStream(connection.getInputStream());
BufferedReader ki=new BufferedReader(new InputStreamReader(System.in));
int flag=0;
System.out.println("connect");
System.out.println("enter the no of frames to be requested to server:");
int c=Integer.parseInt(ki.readLine());
out.write(c);
out.flush();
int i,jj=0;
while(jj<c)
{
i=in.read();
System.out.println("received frame no"+i);
System.out.println("sending acknowledgement for frame no"+i);
out.write(i+1);
out.flush();
j++;
}
out.flush();
in.close();
inn.close();
out.close();
System.out.println("quiting");
}
catch(Exception e)
{
System.out.println(e);
}}}
```

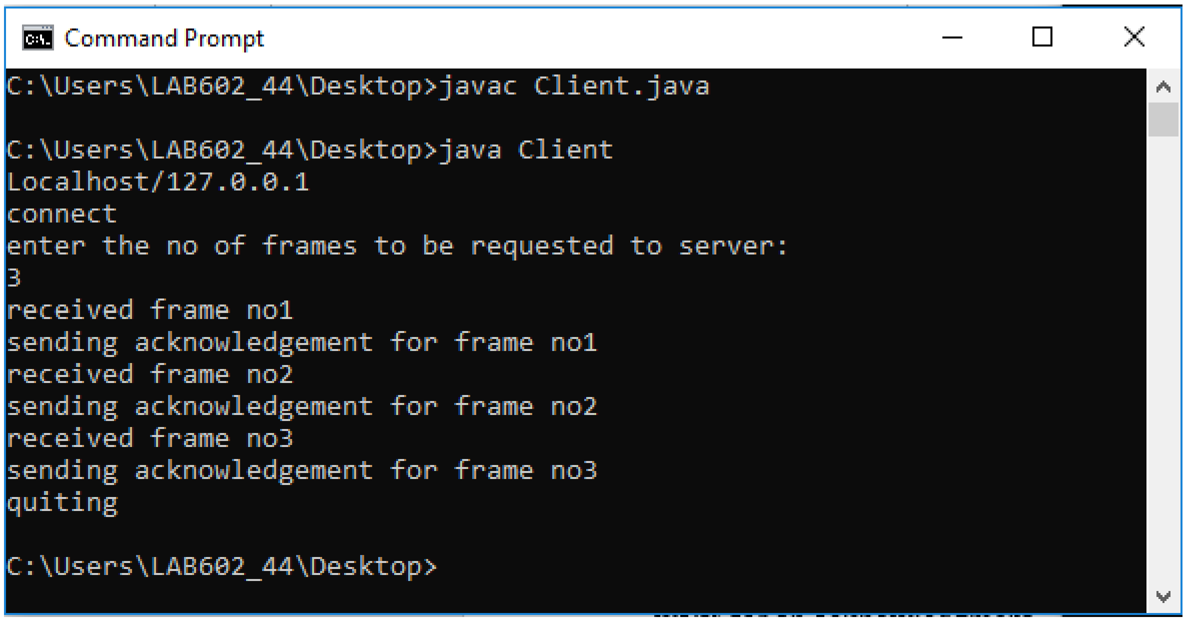
Output:

UDPServer:



```
C:\Users\LAB602_44\Desktop>javac BitServer.java

C:\Users\LAB602_44\Desktop>java BitServer
waiting for connection
received request for sending frames
sending.....
sending frame no1
waiting for acknowledge
received acknowledge for frame no:1as2
sending frame no2
waiting for acknowledge
received acknowledge for frame no:2as3
sending frame no3
waiting for acknowledge
received acknowledge for frame no:3as4
quiting

C:\Users\LAB602_44\Desktop>
```

UDPClient:



```
C:\Users\LAB602_44\Desktop>javac Client.java

C:\Users\LAB602_44\Desktop>java Client
Localhost/127.0.0.1
connect
enter the no of frames to be requested to server:
3
received frame no1
sending acknowledgement for frame no1
received frame no2
sending acknowledgement for frame no2
received frame no3
sending acknowledgement for frame no3
quiting

C:\Users\LAB602_44\Desktop>
```

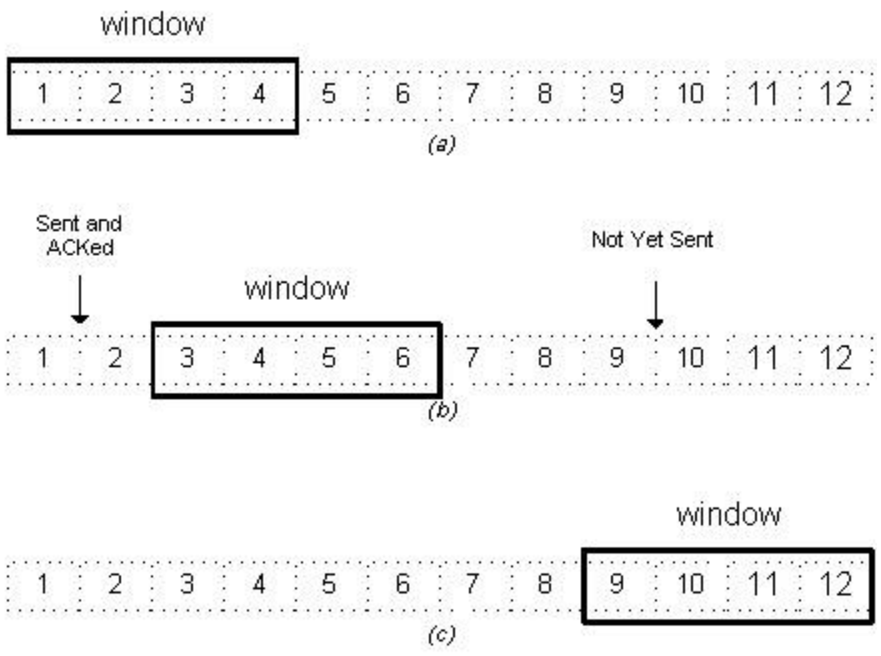
Experiment-7

Aim: Implementation of Sliding Window Protocol.

Description: Sliding window protocol is a method to transmit data on a network. Sliding window protocol is applied on the Data Link Layer of OSI model. At data link layer data is in the form of frames. In Networking, Window simply means a buffer which has data frames that needs to be transmitted.

Both sender and receiver agree on some window size. If window size=w then after sending w frames sender waits for the acknowledgement (ack) of the first frame.

As soon as sender receives the acknowledgement of a frame it is replaced by the next frames to be transmitted by the sender. If receiver sends a collective or cumulative acknowledgement to sender then it understands that more than one frames are properly received, for e.g.: - if ack of frame 3 is received it understands that frame 1 and frame 2 are received properly.



In sliding window protocol, the receiver has to have some memory to compensate any loss in transmission or if the frames are received unordered.

Efficiency of Sliding Window Protocol:

$\eta = (W*t_x)/(t_x+2t_p)$

W = Window Size

t_x = Transmission time
t_p = Propagation delay

Sliding window works in full duplex mode

It is of two types: -

1. Selective Repeat: Sender transmits only that frame which is erroneous or is lost.
2. Go back n: Sender transmits all frames present in the window that occurs after the error bit including error bit also.

Program for Client:

```
import java.io.*;
import java.net.*;
import java.util.*;
public class Slide_Sender
{
    Socket sender;
    ObjectOutputStream out;
    ObjectInputStream in;
    String pkt;
    char data='a';
    int SeqNum = 1, SWS = 5;
    int LAR = 0, LFS = 0;
    int NF;
    Slide_Sender()
    {}
public void SendFrames()
    {
        if((SeqNum<=15)&&(SWS > (LFS - LAR)) )
        {
            try
            {
                NF = SWS - (LFS - LAR);
                for(int i=0;i<NF;i++)
                {
                    pkt = String.valueOf(SeqNum);
                    pkt = pkt.concat(" ");
                    pkt = pkt.concat(String.valueOf(data));
                    out.writeObject(pkt);
                    LFS = SeqNum;
                    System.out.println("Sent  " + SeqNum + "  " + data);
                    data++;
                    if(data=='f')
                        data='a';
                    SeqNum++;
                    out.flush();
                }
            }
            catch(Exception e)
            {}
        }
    }

    public void run() throws IOException
    {
        sender = new Socket("localhost",1500);

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

        while(LAR<15)
        {
            try
            {
                SendFrames();

                String Ack = (String)in.readObject();
                LAR = Integer.parseInt(Ack);
                System.out.println("ack received : " + LAR);
            }
            catch(Exception e)
            {
            }
        }

        in.close();
        out.close();
        sender.close();
        System.out.println("\nConnection Terminated.");
    }

    public static void main(String as[]) throws IOException
    {
        Slide_Sender s = new Slide_Sender();
        s.run();
    }
```

```
}
```

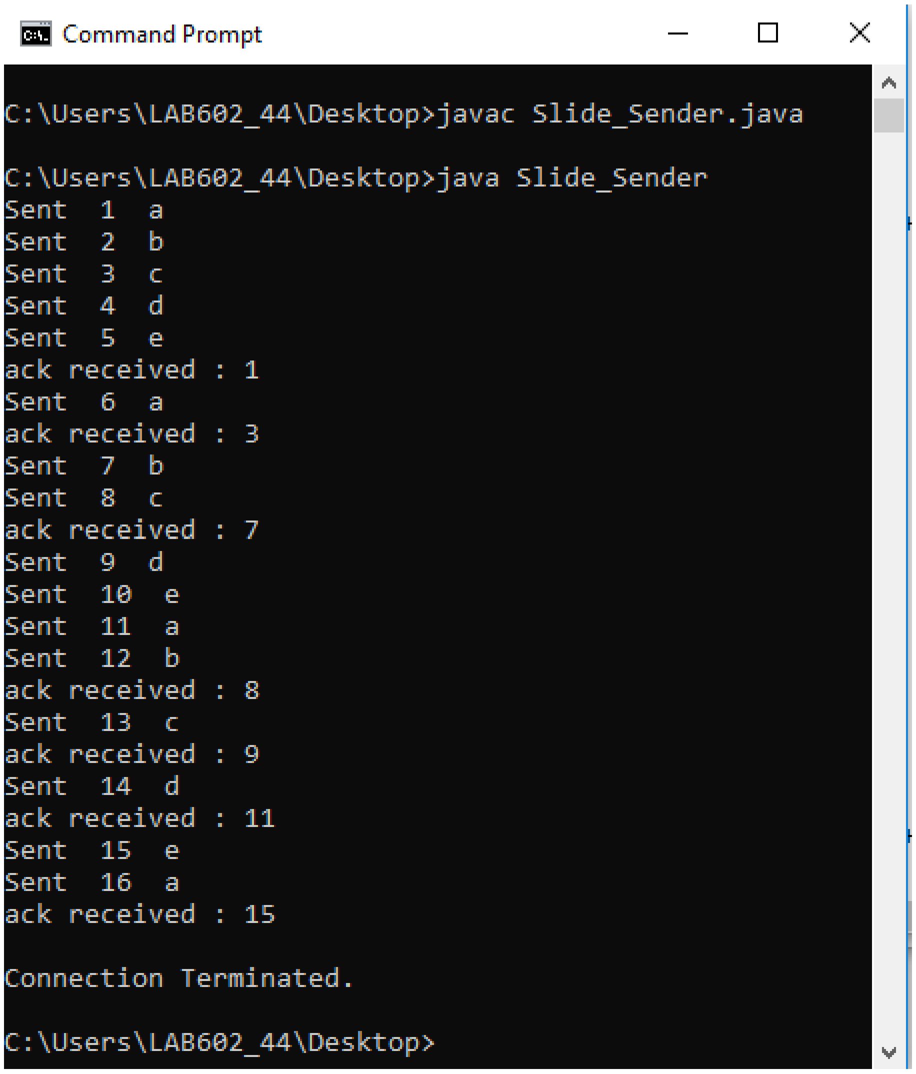
Program for Server:

```
import java.io.*;
import java.net.*;
import java.util.*;
public class Slide_Receiver
{
    ServerSocket reciever;
    Socket conc = null;
    ObjectOutputStream out;
    ObjectInputStream in;
    String ack, pkt, data="";
    int delay ;
    int SeqNum = 0, RWS = 5;
    int LFR = 0;
    int LAF = LFR+RWS;
    Random rand = new Random();
    Slide_Receiver()
    {
    }
    public void run() throws IOException, InterruptedException
    {
        reciever = new ServerSocket(1500,10);
        conc = reciever.accept();
        if(conc!=null)
           System.out.println("Connection established :");
        out = new ObjectOutputStream(conc.getOutputStream());
        in = new ObjectInputStream(conc.getInputStream());
        while(LFR<15)
        {
```

```
try
{
    pkt = (String)in.readObject();
    String []str = pkt.split("\\s");
    ack = str[0];
    data = str[1];
    LFR = Integer.parseInt(ack);
    if((SeqNum<=LFR)||(SeqNum>LAF))
    {
        System.out.println("\nMsg received : "+data);
        delay = rand.nextInt(5);
        if(delay<3 || LFR==15)
        {
            out.writeObject(ack);
            out.flush();
            System.out.println("sending ack " +ack);
            SeqNum++;
        }
        else
            System.out.println("Not sending ack");
    }
    else
    {
        out.writeObject(LFR);
        out.flush();
        System.out.println("resending ack " +LFR);
    }
}
catch(Exception e)
{
}
```

```
        }
        in.close();
        out.close();
        reciever.close();
        System.out.println("\nConnection Terminated.");
    }
    public static void main(String args[]) throws IOException, InterruptedException
    {
        Slide_Receiver R = new Slide_Receiver();
        R.run();
    }
}
```

Output:



```
C:\Users\LAB602_44\Desktop>javac Slide_Sender.java

C:\Users\LAB602_44\Desktop>java Slide_Sender
Sent   1   a
Sent   2   b
Sent   3   c
Sent   4   d
Sent   5   e
ack received : 1
Sent   6   a
ack received : 3
Sent   7   b
Sent   8   c
ack received : 7
Sent   9   d
Sent   10   e
Sent   11   a
Sent   12   b
ack received : 8
Sent   13   c
ack received : 9
Sent   14   d
ack received : 11
Sent   15   e
Sent   16   a
ack received : 15

Connection Terminated.

C:\Users\LAB602_44\Desktop>
```

Experiment-8

Aim: Implementation of Dijkstra Shortest path routing protocol.

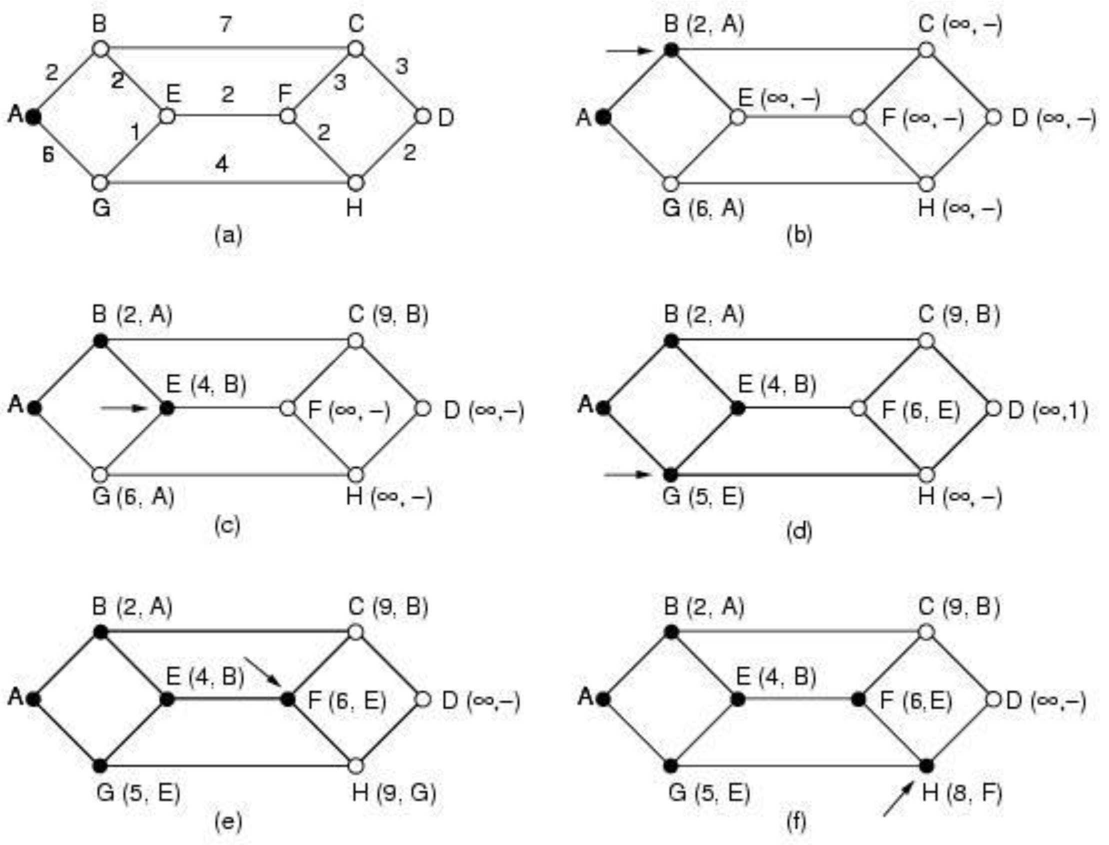
Description: Dijkstra's Algorithm

Dijkstra's algorithm is well-known shortest path routing algorithm. It works on the notion of a candidate neighbouring node set as well as the source's own computation to identify the shortest path to a destination. Another interesting property about Dijkstra's algorithm is that it computes shortest paths to all destinations from a source, instead of just for a specific pair of source and destination nodes at a time; this is very useful, especially in a communication network, since a node wants to compute the shortest path to all destinations.

At the end each node will be labelled with its distance from source node along the best-known path. Initially, no paths are known, so all nodes are labelled with infinity. As the algorithm proceeds and paths are found, the labels may change reflecting better paths. Initially, all labels are tentative. When it is discovered that a label represents the shortest possible path from the source to that node, it is made permanent and never changed thereafter.

Example: Look at the weighted undirected graph of Figure (a), where the weights represent, for example, distance. We want to find shortest path from A to D. We start by making node A as permanent, indicated by a filled in circle. Then we examine each of the nodes adjacent to A (the working node), relabelling each one with the distance to A. Whenever a node is relabelled, we also label it with the node from which the probe was made so that we can construct the final path later. Having examined each of the nodes adjacent to A, we examine all the tentatively labelled nodes in the whole graph and make the one with the smallest label permanent, as shown in Figure (b). This one becomes new working node.

We now start at B, and examine all nodes adjacent to it. If the sum of the label on B and the distance from B to the node being considered is less than the label on the node, we have a shorter path, so the node is relabelled. After all the nodes adjacent to the working node have been inspected and the tentative labels changed if possible, the entire graph is searched for the tentatively labelled node with the smallest value. This node is made permanent and becomes the working node for the next round. The Figure shows the first five steps of the algorithm.



Note: Dijkstra's Algorithm is applicable only when cost of all the nodes is non-negative.

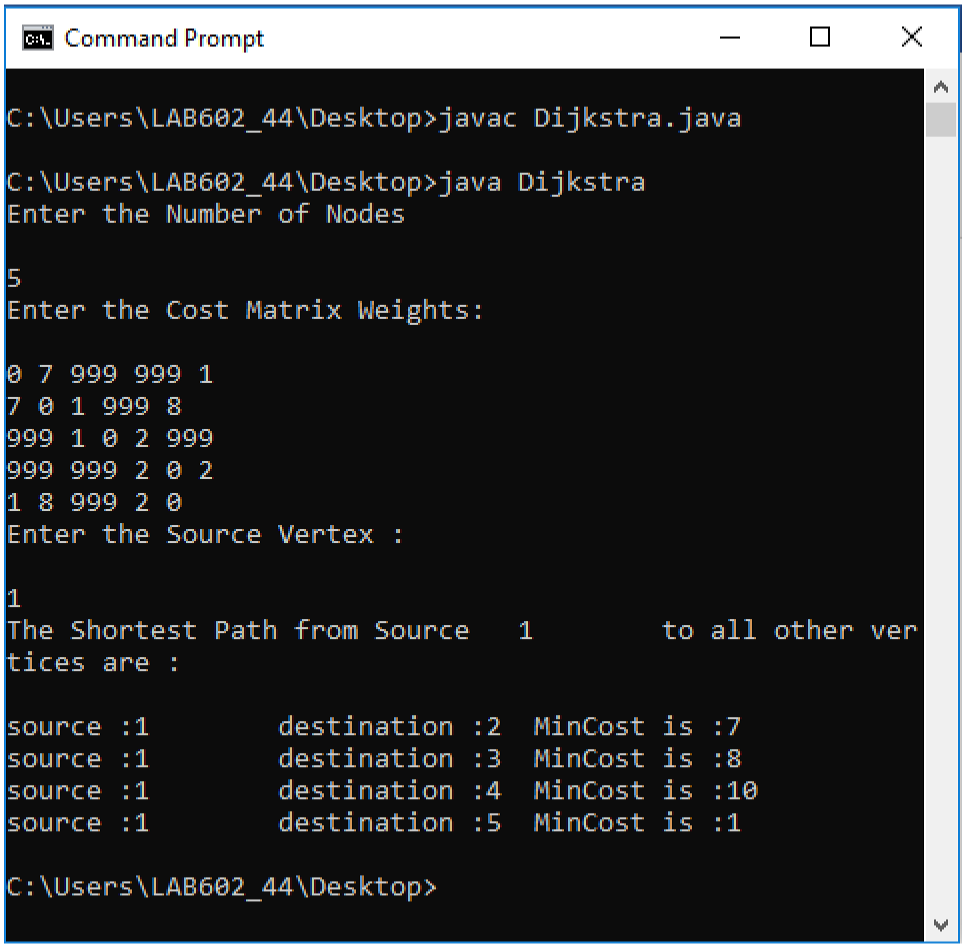
Program:

```
import java.util.*;
public class Dijkstra
{
 public  int distance[] = new int[10];
 public  int cost[][] = new int[10][10];

 public void calc(int n,int s)
```

```
{
 int flag[] = new int[n+1];
 int i,minpos=1,k,c,minimum;

 for(i=1;i<=n;i++)
 {
           flag[i]=0;
     this.distance[i]=this.cost[s][i];
    }
    c=2;
 while(c<=n)
 {
  minimum=99;
  for(k=1;k<=n;k++)
  {
         if(this.distance[k]<minimum && flag[k]!=1)
      {
       minimum=this.distance[i];
       minpos=k;
      }
     }
           flag[minpos]=1;
     c++;
     for(k=1;k<=n;k++)
{
        if(this.distance[minpos]+this.cost[minpos][k] <  this.distance[k] && flag[k]!=1 )
   this.distance[k]=this.distance[minpos]+this.cost[minpos][k];
}
 }

}
```

```
public static void main(String args[])
{
 int nodes,source,i,j;
 Scanner in = new Scanner(System.in);
 System.out.println("Enter the Number of Nodes \n");
 nodes = in.nextInt();
 Dijkstra d = new Dijkstra();
 System.out.println("Enter the Cost Matrix Weights: \n");
       for(i=1;i<=nodes;i++)
         for(j=1;j<=nodes;j++)
   {
           d.cost[i][j]=in.nextInt();
           if(d.cost[i][j]==0)
             d.cost[i][j]=999;
         }
 System.out.println("Enter the Source Vertex :\n");
 source=in.nextInt();

    d.calc(nodes,source);
 System.out.println("The Shortest Path from Source \t"+source+"\t to all other vertices are : \n");
       for(i=1;i<=nodes;i++)
         if(i!=source)
System.out.println("source :"+source+"\t destination :"+i+"\t MinCost is :"+d.distance[i]+"\t");
}
}
```

Output:



Command Prompt

```
C:\Users\LAB602_44\Desktop>javac Dijkstra.java

C:\Users\LAB602_44\Desktop>java Dijkstra
Enter the Number of Nodes

5
Enter the Cost Matrix Weights:

0 7 999 999 1
7 0 1 999 8
999 1 0 2 999
999 999 2 0 2
1 8 999 2 0
Enter the Source Vertex :

1
The Shortest Path from Source   1        to all other ver
tices are :

source :1        destination :2  MinCost is :7
source :1        destination :3  MinCost is :8
source :1        destination :4  MinCost is :10
source :1        destination :5  MinCost is :1

C:\Users\LAB602_44\Desktop>
```

Experiment-9

Aim: Implementation of Distance Vector Routing.

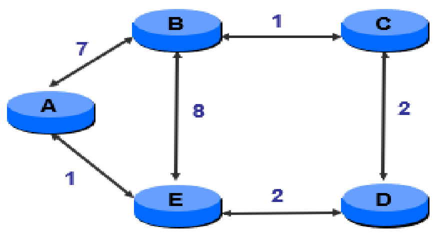
Description: A **distance-vector routing (DVR)** protocol requires that a router inform its neighbours of topology changes periodically. Historically known as the old ARPANET routing algorithm (or known as Bellman-Ford algorithm).

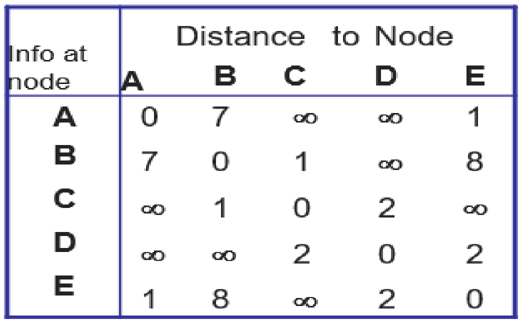
Bellman Ford Basics – Each router maintains a Distance Vector table containing the distance between itself and ALL possible destination nodes. Distances, based on a chosen metric, are computed using information from the neighbours' distance vectors.

Distance Vector Algorithm –

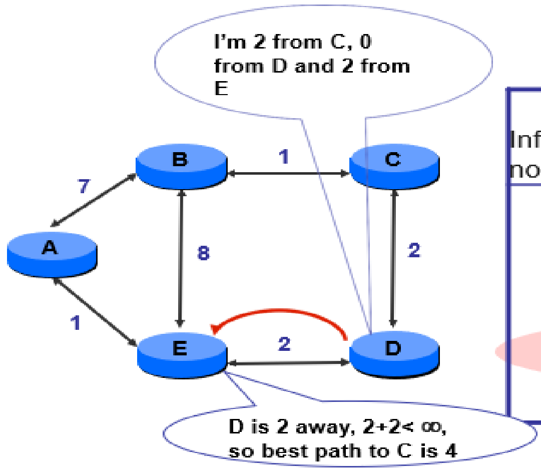
1. A router transmits its distance vector to each of its neighbours in a routing packet.
2. Each router receives and saves the most recently received distance vector from each of its neighbours.
3. A router recalculates its distance vector when:
   - It receives a distance vector from a neighbour containing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

Example:

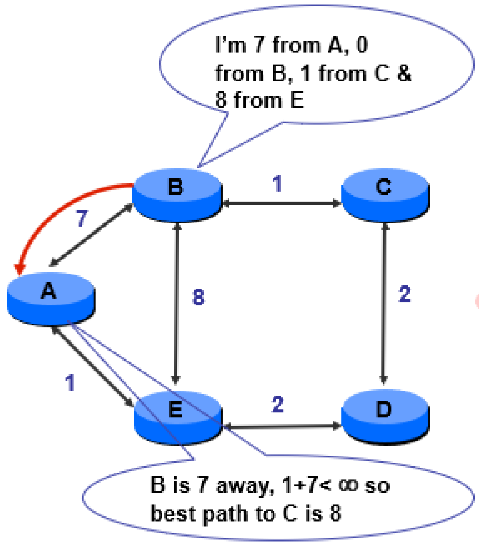




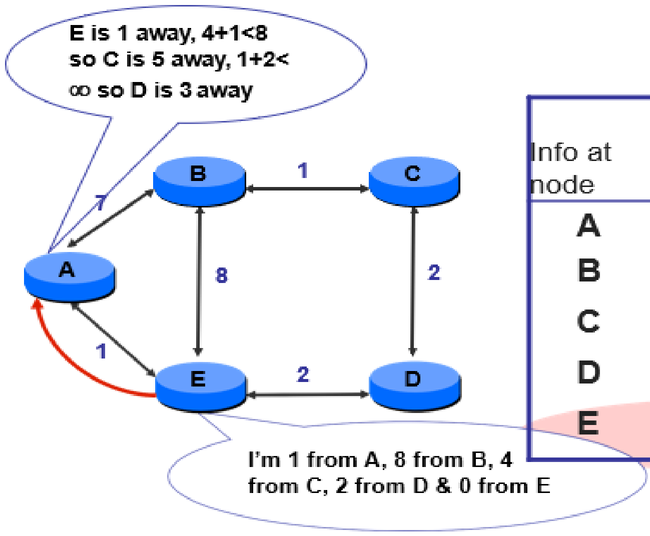
Info at node	Distance to Node				
	A	B	C	D	E
A	0	7	∞	∞	1
B	7	0	1	∞	8
C	∞	1	0	2	∞
D	∞	∞	2	0	2
E	1	8	∞	2	0


I'm 2 from C, 0 from D and 2 from E
D is 2 away, 2+2< ∞, so best path to C is 4
A
B
C
D
E
7
1
8
2
1
2

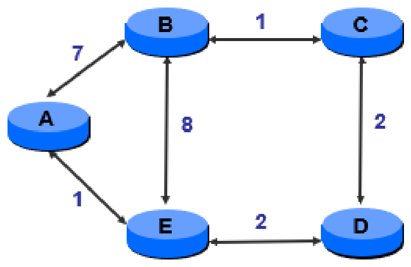
Info at node	Distance to Node				
	A	B	C	D	E
A	0	7	∞	∞	1
B	7	0	1	∞	8
C	∞	1	0	2	∞
D	∞	∞	2	0	2
E	1	8	4	2	0


I'm 7 from A, 0 from B, 1 from C & 8 from E
B is 7 away, 1+7< ∞ so best path to C is 8
A
B
C
D
E
7
1
8
2
1
2

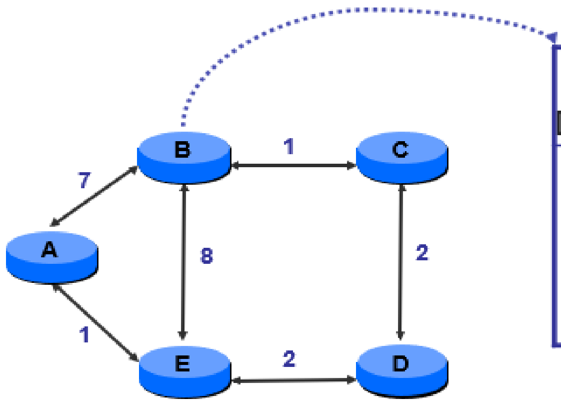
Info at node	Distance to Node				
	A	B	C	D	E
A	0	7	8	∞	1
B	7	0	1	∞	8
C	∞	1	0	2	∞
D	∞	∞	2	0	2
E	1	8	4	2	0


E is 1 away, 4+1<8 so C is 5 away, 1+2< ∞ so D is 3 away
I'm 1 from A, 8 from B, 4 from C, 2 from D & 0 from E
A
B
C
D
E
7
1
8
2
1
2

Info at node	Distance to Node				
	A	B	C	D	E
A	0	7	5	3	1
B	7	0	1	∞	8
C	∞	1	0	2	∞
D	∞	∞	2	0	2
E	1	8	4	2	0



Info at node	Distance To Node A	B	C	D	E
A	0	6	5	3	1
B	6	0	1	3	5
C	5	1	0	2	4
D	3	3	2	0	2
E	1	5	4	2	0



Dest	Next hop A	E	C
A			6
C			1
D			3
E			5

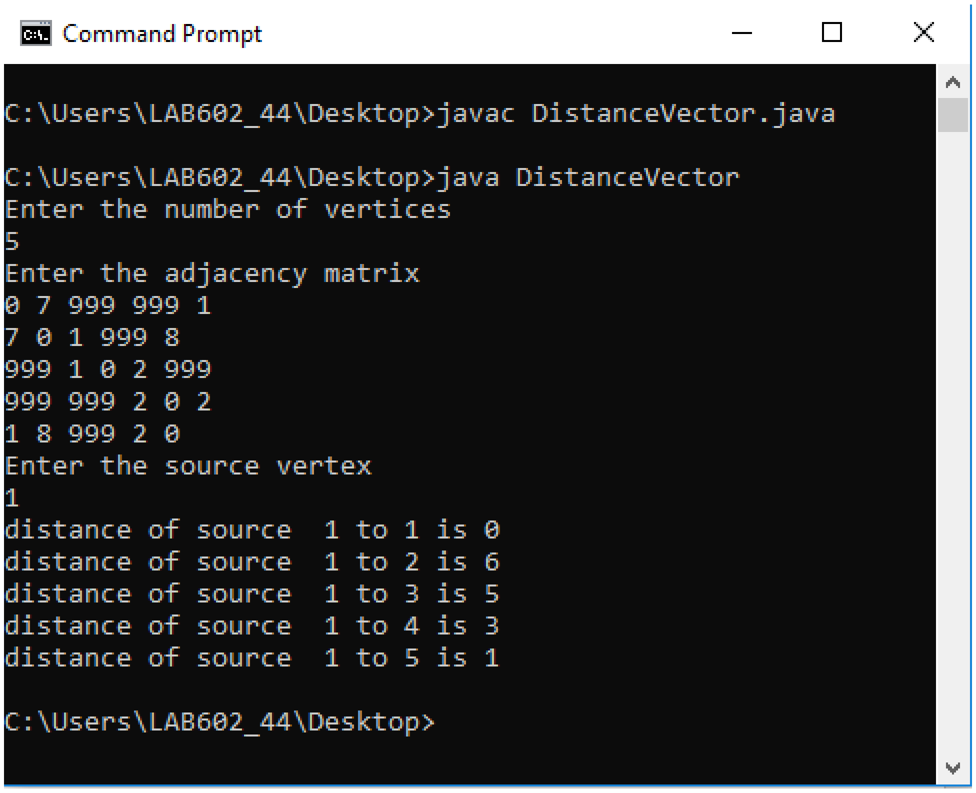
Program:

```
import java.util.Scanner;
public class DistanceVector
   {
        private int distances[];
        private int numberofvertices;
        public static final int MAX_VALUE = 999;
        public DistanceVector(int numberofvertices)
        {
          this.numberofvertices = numberofvertices;
           distances = new int[numberofvertices + 1];
        }
        public void DistanceVectorEvaluation(int source, int adjacencymatrix[][])
        {
           for (int node = 1; node <= numberofvertices; node++)
           {
               distances[node] = MAX_VALUE;
           }
           distances[source] = 0;
           for (int node = 1; node <= numberofvertices - 1; node++)
           {
               for (int sourcenode = 1; sourcenode <= numberofvertices; sourcenode++)
               {
                   for (int destinationnode = 1; destinationnode <= numberofvertices;
destinationnode++)
                   {
                       if (adjacencymatrix[sourcenode][destinationnode]!= MAX_VALUE)
                       {
                           if (distances[destinationnode] >
distances[sourcenode]+adjacencymatrix[sourcenode][destinationnode])
```

```
                        distances[destinationnode] =
distances[sourcenode]+adjacencymatrix[sourcenode][destinationnode];
                  }
               }
            }
         }
         for (int sourcenode = 1; sourcenode <= numberofvertices; sourcenode++)
         {
            for (int destinationnode = 1; destinationnode <= numberofvertices; destinationnode++)
            {
               if (adjacencymatrix[sourcenode][destinationnode] != MAX_VALUE)
               {
                  if (distances[destinationnode] >
distances[sourcenode]+adjacencymatrix[sourcenode][destinationnode])
                     System.out.println("The Graph contains negative egde cycle");
               }
            }
         }
         for (int vertex = 1; vertex <= numberofvertices; vertex++)
         {
            System.out.println("distance of source " + source + " to "+ vertex + " is " +
distances[vertex]);
         }
      }
      public static void main(String... arg)
      {
         int numberofvertices = 0;
         int source;
         Scanner scanner = new Scanner(System.in);
         System.out.println("Enter the number of vertices");
         numberofvertices = scanner.nextInt();
```

```
        int adjacencymatrix[][] = new int[numberofvertices + 1][numberofvertices + 1];
        System.out.println("Enter the adjacency matrix");
        for (int sourcenode = 1; sourcenode <= numberofvertices; sourcenode++)
        {
            for (int destinationnode = 1; destinationnode <= numberofvertices; destinationnode++)
            {
                adjacencymatrix[sourcenode][destinationnode] = scanner.nextInt();
                 if (sourcenode == destinationnode)
                 {
                     adjacencymatrix[sourcenode][destinationnode] = 0;
                     continue;
                 }
                 if (adjacencymatrix[sourcenode][destinationnode] == 0)
                 {
                     adjacencymatrix[sourcenode][destinationnode] = MAX_VALUE;
                 }
            }
        }
        System.out.println("Enter the source vertex");
        source = scanner.nextInt();
        DistanceVector bellmanford = new DistanceVector(numberofvertices);
        bellmanford.DistanceVectorEvaluation(source, adjacencymatrix);
        scanner.close();
    }
}
```

Output:



```
C:\Users\LAB602_44\Desktop>javac DistanceVector.java

C:\Users\LAB602_44\Desktop>java DistanceVector
Enter the number of vertices
5
Enter the adjacency matrix
0 7 999 999 1
7 0 1 999 8
999 1 0 2 999
999 999 2 0 2
1 8 999 2 0
Enter the source vertex
1
distance of source  1 to 1 is 0
distance of source  1 to 2 is 6
distance of source  1 to 3 is 5
distance of source  1 to 4 is 3
distance of source  1 to 5 is 1

C:\Users\LAB602_44\Desktop>
```

Experiment-10

Aim: Implementation of ECHO Command.

Description: Often referred to as an **echo** check, **echo** describes when data is sent to a **computer** or other **network** devices, and that information is sent back to verify the information was received. **Echo** can be a **command** used with operating systems, **network** devices, or a function of a device.

ECHOSERVER:

1. Start the program.
2. Import.net and other necessary packages.
3. Declare objects for DataInputStream, Socket and PrintWriter to receive server message and send it back.
4. Store the message in a string and print the message using print() method.
5. Send the same received message to the server using PrintWriter and socket.
6. When the received message is end, then exit the program execution.

ECHOCLIENT:

1. Start the program.
2. Import.net and other necessary packages.
3. Declare objects for ServerSocket and Socket to send the message.
4. Declare objects for PrintStream to write message from server to client.
5. Get the user input and store it in a string.
6. Print the string in the socket using printStream to be received by the receiver.
7. Using the Print () method, receive the client echo message and print it.
8. If the message is end, terminate the program and exit.

Program for ECHOSERVER:

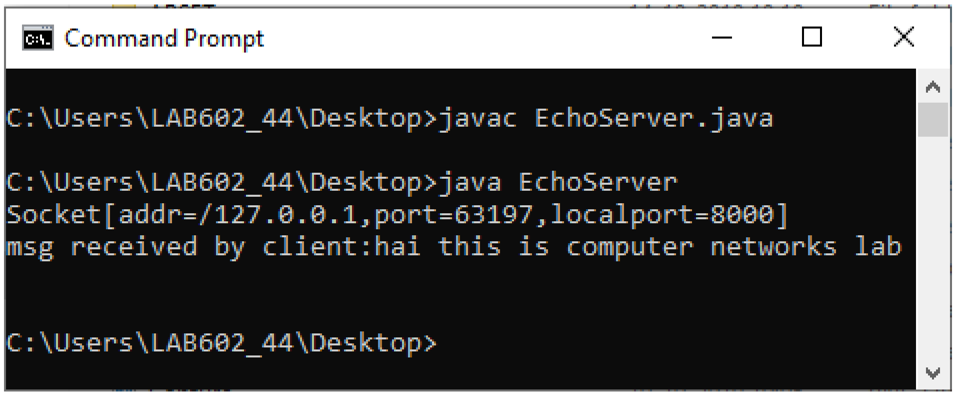
```
import java.io.*;
import java.net.*;
import java.util.Scanner;
class echoserver
{
public static void main(String args[])
{
try
{
Socket s=null;
ServerSocket ss=new ServerSocket(8000);
s=ss.accept();
System.out.println(s);
BufferedReader br=new BufferedReader(new InputStreamReader(s.getInputStream()));
PrintWriter print=new PrintWriter(s.getOutputStream());
int i=1;
while(i>0)
{
String str=br.readLine();
if(str.equals("."))
break;
System.out.println("msg received by client:"+str);
print.println(str);
print.flush();
}}
catch(IOException e)
{}
}
}
```

Program for ECHOCLIENT:

```
import java.io.*;
import java.net.*;
class echoclient
{
public static void main(String a[])
{
try
{
Socket s=new Socket("LocalHost",8000);
DataInputStream in=new DataInputStream(System.in);
BufferedReader br1=new BufferedReader(new InputStreamReader(System.in));
BufferedReader br2=new BufferedReader(new InputStreamReader(s.getInputStream()));
PrintWriter print=new PrintWriter(s.getOutputStream());
System.out.println("\n msg to be echo:");
String str=br1.readLine();
print.println(str);
print.flush();
System.out.println(br2.readLine());
}
catch(UnknownHostException e)
{}
catch(IOException e)
{
System.out.println("\n error:"+e);
}
}
}
```

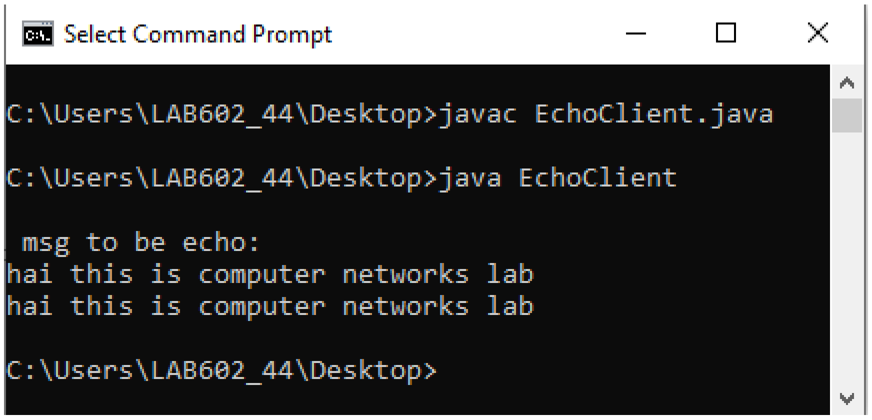
Output:

Echoserver:



```
C:\Users\LAB602_44\Desktop>javac EchoServer.java

C:\Users\LAB602_44\Desktop>java EchoServer
Socket[addr=/127.0.0.1,port=63197,localport=8000]
msg received by client:hai this is computer networks lab


C:\Users\LAB602_44\Desktop>
```

EchoClient:



```
C:\Users\LAB602_44\Desktop>javac EchoClient.java

C:\Users\LAB602_44\Desktop>java EchoClient

 msg to be echo:
hai this is computer networks lab
hai this is computer networks lab

C:\Users\LAB602_44\Desktop>
```

Experiment-11

Aim: Java program for PING "Round trip time (RTT)"

Description: Ping is a computer network administration software utility used to test the reachability of a host on an Internet Protocol network. It is available for virtually all operating systems that have networking capability, including most embedded network administration software.

ALGORITHM:

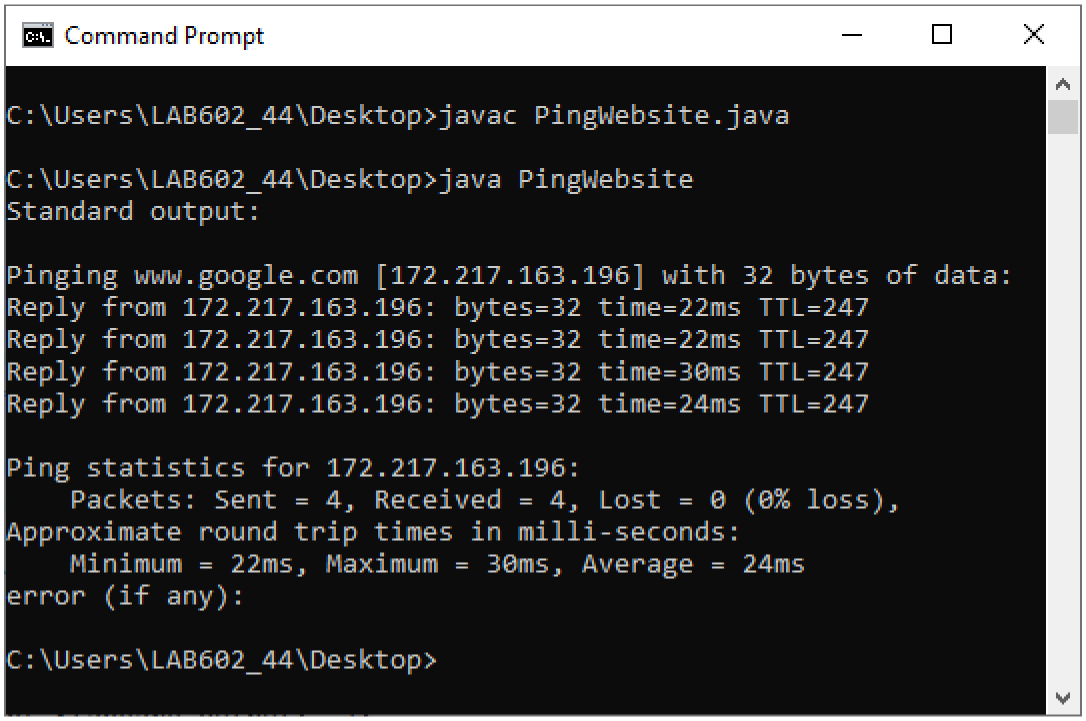
1. Start the program.
2. Import.net and other necessary packages.
3. Initialize the ping server with both sockets as null value.
4. Start the serversocket.
5. At the client and give the IP address of the server.
6. The client program is then started by starting socket.
7. At the receiver end, the server is pinged.

Program:

```
// Java program for ping using sub-process
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.ArrayList;
public class PingWebsite
{
        // method for finding the ping statics of website
        static void commands(ArrayList<String> commandList)
                                                            throws Exception
        {
                // creating the sub process, execute system command
                ProcessBuilder build = new ProcessBuilder(commandList);
                Process process = build.start();
```

```
            // to read the output
            BufferedReader input = new BufferedReader(new InputStreamReader
                                                    (process.getInputStream()));
            BufferedReader Error = new BufferedReader(new InputStreamReader
                                                    (process.getErrorStream()));
            String s = null;

            System.out.println("Standard output: ");
            while((s = input.readLine()) != null)
            {
                System.out.println(s);
            }
            System.out.println("error (if any): ");
            while((s = Error.readLine()) != null)
            {
                System.out.println(s);
            }
        }
        // Driver method
        public static void main(String args[]) throws Exception
        {
            // creating list for commands
            ArrayList<String> commandList = new ArrayList<String>();

            commandList.add("ping");
            // can be replaced by IP
            commandList.add("www.google.com");

            PingWebsite.commands(commandList);
        }
    }
```

Output:



```
C:\Users\LAB602_44\Desktop>javac PingWebsite.java

C:\Users\LAB602_44\Desktop>java PingWebsite
Standard output:

Pinging www.google.com [172.217.163.196] with 32 bytes of data:
Reply from 172.217.163.196: bytes=32 time=22ms TTL=247
Reply from 172.217.163.196: bytes=32 time=22ms TTL=247
Reply from 172.217.163.196: bytes=32 time=30ms TTL=247
Reply from 172.217.163.196: bytes=32 time=24ms TTL=247

Ping statistics for 172.217.163.196:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 22ms, Maximum = 30ms, Average = 24ms
error (if any):

C:\Users\LAB602_44\Desktop>
```

Experiment-12

Aim: Implementation of TRACEROUTE Command.

Description: **Traceroute** is a utility that records the route (the specific gateway **computers** at each hop) through the Internet between your **computer** and a specified destination **computer**. It also calculates and displays the amount of time each hop took.

The traceroute command is used to discover the routes that packets actually take when traveling to their destination. The device (for example, a router or a PC) sends out a sequence of User Datagram Protocol (UDP) datagrams to an invalid port address at the remote host.

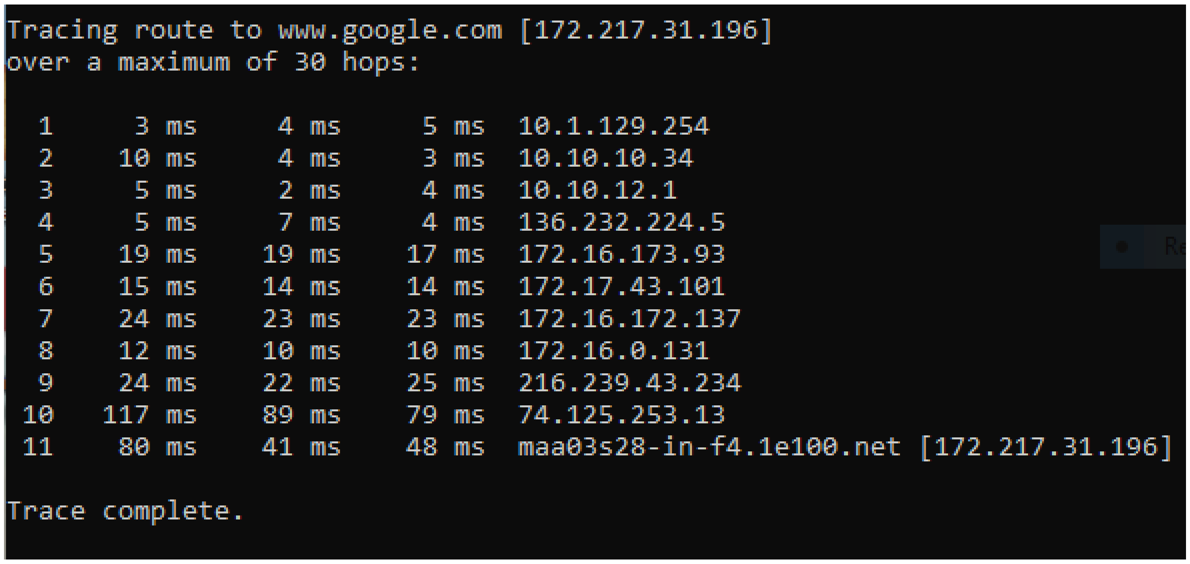
Three datagrams are sent, each with a Time-To-Live (TTL) field value set to one. The TTL value of 1 causes the datagram to "timeout" as soon as it hits the first router in the path; this router then responds with an ICMP Time Exceeded Message (TEM) indicating that the datagram has expired.

Another three UDP messages are now sent, each with the TTL value set to 2, which causes the second router to return ICMP TEMs. This process continues until the packets actually reach the other destination. Since these datagrams are trying to access an invalid port at the destination host, ICMP Port Unreachable Messages are returned, indicating an unreachable port; this event signals the Traceroute program that it is finished. The purpose behind this is to record the source of each ICMP Time Exceeded Message to provide a trace of the path the packet took to reach the destination.

Program:

```
import java.io.*;
import java.net.*;
import java.lang.*;
class Traceroute
{
   public static void main(String args[]){
   BufferedReader in;
      try{
         Runtime r   =   Runtime.getRuntime();
         Process p   =   r.exec("tracert www.google.com");
         in  =   new BufferedReader(new InputStreamReader(p.getInputStream()));
         String line;
         if(p==null)
            System.out.println("could not connect");
         while((line=in.readLine())!=null)
            System.out.println(line);
            //in.close();
         }
      }catch(IOException e){
      System.out.println(e.toString());
      }
   }
}
```

Output:



```
Tracing route to www.google.com [172.217.31.196]
over a maximum of 30 hops:

  1     3 ms     4 ms     5 ms  10.1.129.254
  2    10 ms     4 ms     3 ms  10.10.10.34
  3     5 ms     2 ms     4 ms  10.10.12.1
  4     5 ms     7 ms     4 ms  136.232.224.5
  5    19 ms    19 ms    17 ms  172.16.173.93
  6    15 ms    14 ms    14 ms  172.17.43.101
  7    24 ms    23 ms    23 ms  172.16.172.137
  8    12 ms    10 ms    10 ms  172.16.0.131
  9    24 ms    22 ms    25 ms  216.239.43.234
 10   117 ms    89 ms    79 ms  74.125.253.13
 11    80 ms    41 ms    48 ms  maa03s28-in-f4.1e100.net [172.217.31.196]

Trace complete.
```

Experiment-13

Aim: Write a java Program to implement Subnet Mask.

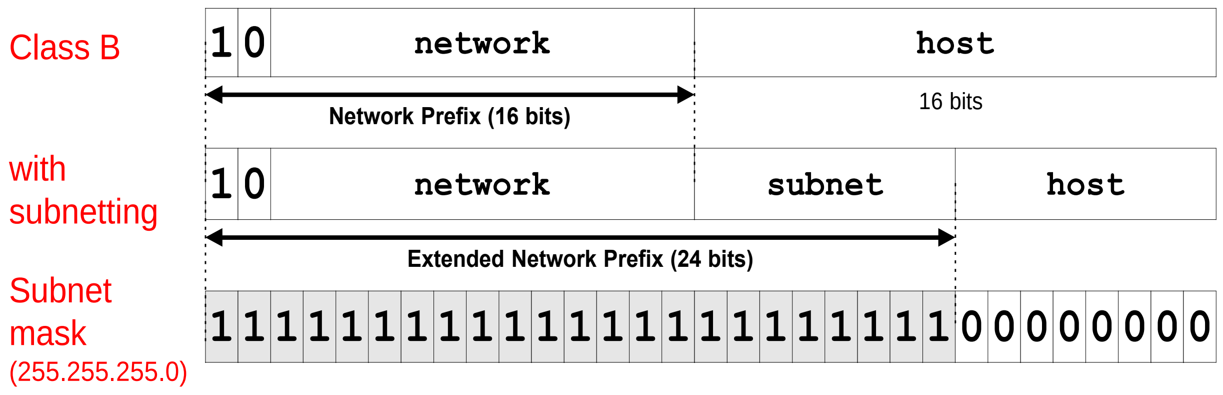
Description: A subnet mask is a number that defines a range of IP addresses available within a network. A single subnet mask limits the number of valid IPs for a specific network. Multiple subnet masks can organize a single network into smaller networks (called subnetworks or subnets). Systems within the same subnet can communicate directly with each other, while systems on different subnets must communicate through a router.

A subnet mask hides (or masks) the network part of a system's IP address and leaves only the host part as the machine identifier. It uses the same format as an IPv4 address — four sections of one to three numbers, separated by dots. Each section of the subnet mask can contain a number from 0 to 255, just like an IP address.

For example, a typical subnet mask for a Class C IP address is:**255.255.255.0**

Subnet Masks:

- Routers and hosts use an extended network prefix (subnet mask) to identify the start of the host numbers



* There are different ways of subnetting. Commonly used netmasks for university networks with /16 prefix (Class B) are 255.255.255.0 and 255.255.0.0

Advantages of Subnetting:

- With subnetting, IP addresses use a 3-layer hierarchy:

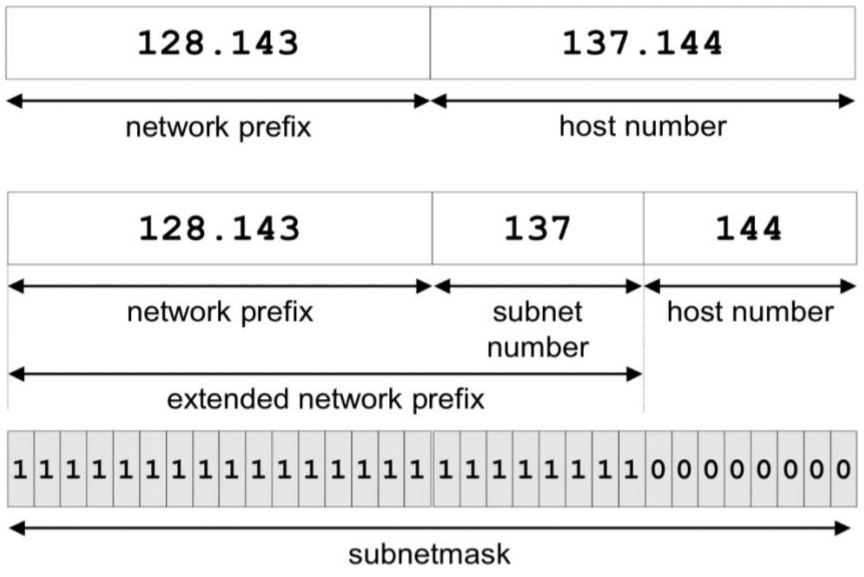
Network

Subnet

Host

- Improves efficiency of IP addresses by not consuming an entire address space for each physical network.
- Reduces router complexity. Since external routers do not know about subnetting, the complexity of routing tables at external routers is reduced.
- Note: Length of the subnet mask need not be identical at all subnetworks.

Subnetting Example:



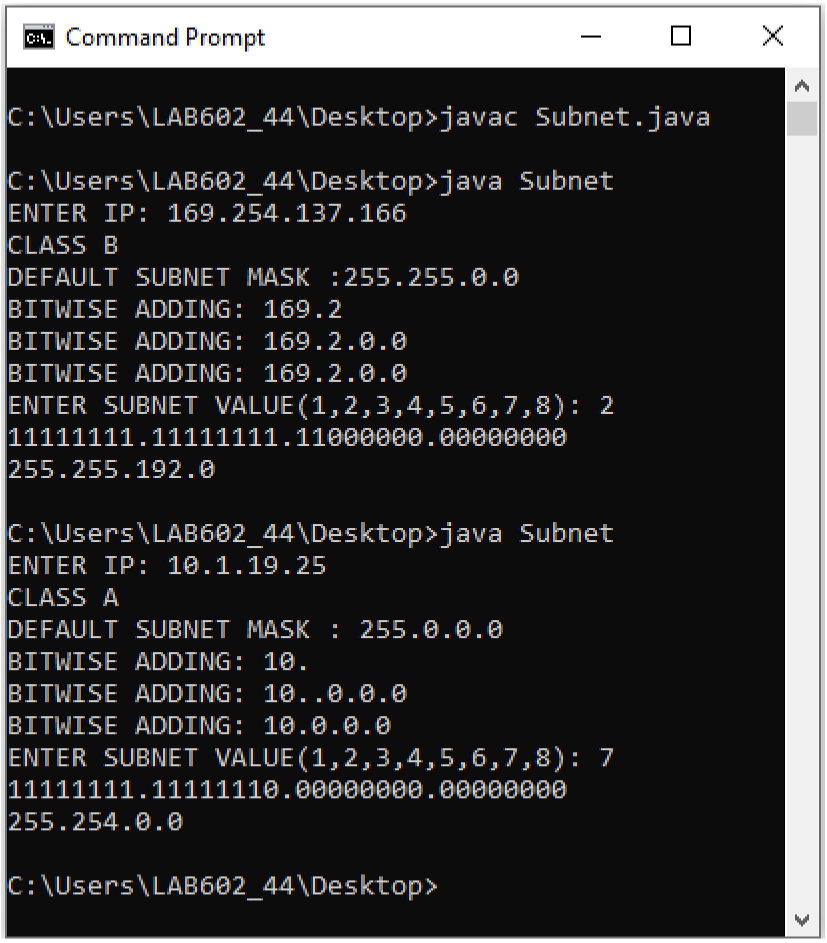
Program:

```
import java.io.*;
import java.util.*;
public class Subnet {

  static String sq = "";
  public static void main(String[] args) throws IOException {
   InputStreamReader isr = new InputStreamReader(System.in);
   BufferedReader br = new BufferedReader(isr);
   int x=0;
   String s,s1,sx;
   System.out.print("ENTER IP: ");
   s= br.readLine();
   s1 = s.substring(0,3);
   s1 = s1.replace(".","");
   int a = Integer.parseInt(s1);
   if(a>=0 && a<=127){
        String dm = "255.0.0.0";
        x=1;
        System.out.println("CLASS A");
        System.out.println("DEFAULT SUBNET MASK : " +dm);
   }else if (a>=128 && a<=191){
             String dm = "255.255.0.0";
             x=2;
             System.out.println("CLASS B");
             System.out.println("DEFAULT SUBNET MASK :" +dm);
         }else if(a>=192 && a<=223){
                  String dm = "255.255.255.0";
                  x=3;
                  System.out.println("CLASS C");
                  System.out.println("DEFAULT SUBNET MASK : " +dm);
                 }
               else System.out.println("OUT OF RANGE");
assert x != 0 : "PROGRAM CANNOT RUN FURTHER ";
    if(x==1){
      sx = s.substring(0,3);
System.out.println("BITWISE ADDING: " +sx);
      sx = sx.concat(".0.0.0");
System.out.println("BITWISE ADDING: " +sx);
      sx = sx.replace("...","..");
      sx = sx.replace("..",".");
      System.out.println("BITWISE ADDING: " +sx);
   }else if(x==2){
           sx = s.substring(0,5);
System.out.println("BITWISE ADDING: " +sx);
           sx = sx.concat(".0.0");
System.out.println("BITWISE ADDING: " +sx);
```

```
          sx = sx.replace("...","..");
          sx = sx.replace("..",".");
          System.out.println("BITWISE ADDING: " +sx);
         }else if(x==3){
               sx = s.substring(0,7);
System.out.println("BITWISE ADDING: " +sx);
               sx = sx.concat(".0");
System.out.println("BITWISE ADDING: " +sx);
               sx = sx.replace("...","..");
               sx = sx.replace("..",".");
               System.out.println("BITWISE ADDING: " +sx);

              }
     System.out.print("ENTER SUBNET VALUE(1,2,3,4,5,6,7,8): ");
       int z = Integer.parseInt(br.readLine());
       convert(x,z);
  }

   static void convert(int x,int z){
    int count=0;
      while(z!=0){
       sq = sq.concat("1");
       z--;
       count++;
      }

      while(count!=8){
       sq = sq.concat("0");
       count++;
      }
    int decimal=Integer.parseInt(sq,2);
```

```
      if(x==1){
        System.out.println("11111111." + sq + ".00000000.00000000");
        System.out.println("255." + decimal + ".0.0");
      }
      if(x==2){
          System.out.println("11111111.11111111." + sq + ".00000000");
          System.out.println("255.255." + decimal + ".0");
      }
      if(x==3){
          System.out.println("11111111.11111111.11111111." + sq + "");
           System.out.println("255.255.255." + decimal);
      }
 }
}
```

Output:



```
C:\Users\LAB602_44\Desktop>javac Subnet.java

C:\Users\LAB602_44\Desktop>java Subnet
ENTER IP: 169.254.137.166
CLASS B
DEFAULT SUBNET MASK :255.255.0.0
BITWISE ADDING: 169.2
BITWISE ADDING: 169.2.0.0
BITWISE ADDING: 169.2.0.0
ENTER SUBNET VALUE(1,2,3,4,5,6,7,8): 2
11111111.11111111.11000000.00000000
255.255.192.0

C:\Users\LAB602_44\Desktop>java Subnet
ENTER IP: 10.1.19.25
CLASS A
DEFAULT SUBNET MASK : 255.0.0.0
BITWISE ADDING: 10.
BITWISE ADDING: 10..0.0.0
BITWISE ADDING: 10.0.0.0
ENTER SUBNET VALUE(1,2,3,4,5,6,7,8): 7
11111111.11111110.00000000.00000000
255.254.0.0

C:\Users\LAB602_44\Desktop>
```