ANNAMACHARYA INSTITUTE OF TECHNOLOGY AND SCIENCES (AUTONOMOUS)

TIRUPATI

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COMPUTER NETWORKS LABORATORY(19APC0525)

LAB MANUAL

(Regulation AK19)

Prepared by, **Approved by,**

LIST OF EXPERIMENTS & SCHEDULE

COURSE CODE: 19APC0525
COURSE TITLE: COMPUTER NETWORKS LAB

Exp. No.	Title	Week No.
1	IMPLEMENTATION OF ERROR DETECTION AND ERROR CORRECTION TECHNIQUES	1
2	BIT STUFFING PROGRAM	2
3	STUDY OF NETWORK DEVICES IN DETAIL.	3
4	IMPLEMENT CRC-CCITT POLYNOMIAL TO OBTAIN CRC CODE.	4
5	IMPLEMENT STOP AND WAIT PROTOCOL AND SLIDING WINDOW PROTOCOL	5
6	IMPLEMENT DIJKSTRA‘ S ALGORITHM TO COMPUTE THE SHORTEST ROUTING PATH	6
7	DISTANCE VECTOR ROUTING.	7
8	SOCKET PROGRAM FOR ECHO.	8
9	CLIENT- SERVER APPLICATION FOR CHAT	9
10	CASE STUDY ON ROUTING ALGORITHMS	10

Course Coordinator **HOD**

IMPLEMENTATION OF ERROR DETECTION AND ERROR CORRECTION TECHNIQUES

AIM:

To implement error detection and error correction techniques.

SOFTWARE REQUIREMENTS:

Turbo C

THEORY:

The upper layers work on some generalized view of network architecture and are not aware of actual hardware data processing. Hence, the upper layers expect error-free transmission between the systems. Most of the applications would not function expectedly if they receive erroneous data. Applications such as voice andvideo may not be that affected and with some errors they may still function well. Data-link layer uses some error control mechanism to ensure that frames (data bit streams) are transmitted with certain level of accuracy. But to understand how errors is controlled, it is essential to know what types of errors may occur. CRC is a different approach to detect if the received frame contains valid data. This technique involves binary division of the data bits being sent. The divisor is generated using polynomials. The sender performs a division operation on the bits being sent and calculates the remainder. Before sending the actual bits, the sender adds the remainder at the end of the actual bits. Actual data bits plus the remainder is called a codeword. The sendertransmits data bits as codewords.

ALGORITHM:

1. Open Turbo c++ software and type the program for error detection
2. Get the input in the form of bits.
3. Append 16 zeros as redundancy bits.
4. Divide the appended data using a divisor polynomial.
5. The resulting data should be transmitted to the receiver.
6. At the receiver the received data is entered.
7. The same process is repeated at the receiver.
8. If the remainder is zero there is no error otherwise there is some error in the received bits
9. Run the program.

PROGRAM

```
#include<stdio.h>

char m[50],g[50],r[50],q[50],temp[50]; void caltrans(int);

void crc(int); void calram(); void shiftl(); int main()

{

int n,i=0;

char ch,flag=0;

printf("Enter the frame bits:"); while((ch=getc(stdin))!='\n') m[i++]=ch;

n=i; for(i=0;i<16;i++) m[n++]='0';
```

```
m[n]='\0';
printf("Message after appending 16 zeros:%s",m); for(i=0;i<=16;i++)
g[i]='0'; g[0]=g[4]=g[11]=g[16]='1';g[17]='\0';
printf("\ngenerator:%s\n",g); crc(n);
printf("\n\nquotient:%s",q); caltrans(n);
printf("\ntransmitted frame:%s",m); printf("\nEnter transmitted frame:"); scanf("\n%s",m);
printf("CRC checking\n"); crc(n);
printf("\n\nlast remainder:%s",r); for(i=0;i<16;i++)
if(r[i]!='0') flag=1; else continue; if(flag==1)
printf("Error during transmission"); else
printf("\n\nReceived freme is correct");
}
void crc(int n)
{
int i,j; for(i=0;i<n;i++) temp[i]=m[i]; for(i=0;i<16;i++) r[i]=m[i];
printf("\nintermediate remainder\n"); for(i=0;i<n-16;i++)
{ if(r[0]=='1')
{ q[i]='1';
calram();
}
else
{ q[i]='0';
shiftl();
} r[16]=m[17+i];
r[17]='\0';
printf("\nremainder %d:%s",i+1,r); for(j=0;j<=17;j++)
temp[j]=r[j];
}
q[n-16]='\0';
}
void calram()
```

```
{
int i,j; for(i=1;i<=16;i++)
r[i-1]=((int)temp[i]-48)^((int)g[i]-48)+48;
}
void shiftl()
{
int i; for(i=1;i<=16;i++) r[i-1]=r[i];
}
void caltrans(int n)
{
int i,k=0;
for(i=n-16;i<n;i++)
m[i]=((int)m[i]-48)^((int)r[k++]-48)+48; m[i]='\0';
}
```

OUTPUT:

Enter the Frame Bits:

1011
The msg after appending 16 zeros:

10110000000000000000
The Transmitted frame is:**10111011000101101011**Enter the transmitted Frame
10111011000101101011

Received msg:**10111011000101101011**
1The Remainder is:**0000000000000000**
Received frame is correct.

VIVA QUESTIONS:

1. What are the types of errors?
2. What is Error Detection? What are its methods?
3. What is Redundancy?
4. What is CRC?
5. What is Checksum?

BIT STUFFING PROGRAM IN C

Aim:

To implement Bit Stuffing in C Language.

Description:

Bit stuffing refers to the insertion of one or more bits into a data transmission as a way to provide signaling information to a receiver. The receiver knows how to detect, remove or disregard the stuffed bits.

In the data link layer of the Open Systems Interconnection model, a stream of bits is divided into more manageable units, or frames. Each frame contains the sending and receiving information to facilitate transmission.

To separate the frames, an 8-bit flag byte is injected at the beginning and end of the sequence. This keeps the receiver from interpreting the flag as part of the transmitted information.

HDLC packets

In another example of bit stuffing, a standard High-Level Data Link Control (HDLC) packet begins and ends with 01111110. To make sure this sequence doesn't reappear before the end of the packet, a 0 is inserted after every five consecutive 1s.

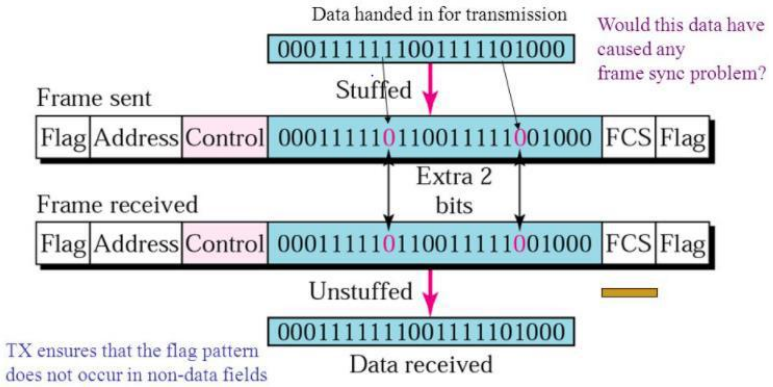


Fig: Bit stuffing example

Algorithm:

1. Start
2. Initialize the array for transmitted stream with the special bit pattern 0111 1110 which indicates the beginning of the frame.
3. Get the bit stream to be transmitted in to the array.
4. Check for five consecutive ones and if they occur, stuff a bit 0
5. Display the data transmitted as it appears on the data line after appending 0111 1110 at the end
6. For de–stuffing, copy the transmitted data to another array after detecting the stuffed bits
7. Display the received bit stream
8. Stop

Program:

```
#include<stdio.h>
#include<string.h>
int main()
{
   int a[20],b[30],i,j,k,count,n;
   printf("Enter frame size (Example: 8):");
   scanf("%d",&n);
   printf("Enter the frame in the form of 0 and 1 :");
   for(i=0; i<n; i++)
      scanf("%d",&a[i]);
   i=0;
   count=1;
   j=0;
   while(i<n)
   {
      if(a[i]==1)
      {
         b[j]=a[i];
         for(k=i+1; a[k]==1 && k<n && count<5; k++)
         {
            j++;
            b[j]=a[k];
            count++;
            if(count==5)
            {
               j++;
               b[j]=0;
            }
            i=k;
         }
      }
      else
      {
         b[j]=a[i];
      }
      i++;
 j++;
   }
   printf("After Bit Stuffing :");
   for(i=0; i<j; i++)
      printf("%d",b[i]);
   return 0;
}
```

OUTPUT for BIT STUFFING:

```
Enter frame size (Example: 8):12
Enter the frame in the form of 0 and 1 :0 1 0 1 1 1 1 1 1 0 0 1
After Bit Stuffing :0101111101001
```

PROGRAM TO IMPLEMENT CHARACTER STUFFING

Aim:

To implement Character Stuffing in C Language.

Algorithm for Character stuffing

1. Start
2. Append DLE STX at the beginning of the string
3. Check the data if character is present; if character DLE is present in the string (example DOODLE) insert another DLE in the string (ex: DOODLEDLE)
4. Transmit DLE ETXat the end of the string
5. Display the string
6. Stop

Description:

Same idea as bit-stuffing, but operates on bytes instead of bits.

Use reserved characters to indicate the start and end of a frame. For instance, use the two-character sequence DLE STX (Data-Link Escape, Start of TeXt) to signal the beginning of a frame, and the sequence DLE ETX (End of TeXt) to flag the frame's end.

Problem: What happens if the two-character sequence DLE ETX happens to appear in the frame itself?

Solution: Use *character stuffing*; within the frame, replace every occurrence of DLE with the two-character sequence DLE DLE. The receiver reverses the processes, replacing every occurrence of DLE DLE with a single DLE.

Example: If the frame contained ``A B DLE D E DLE", the characters transmitted over the channel would be ``DLE STX A B DLE DLE D E DLE DLE DLE ETX".

Program

```
#include<stdio.h>
#include<string.h>
main()
{
	char a[30], fs[50] = " ", t[3], sd, ed, x[3], s[3], d[3], y[3];
	int i, j, p = 0, q = 0;
	clrscr();
	printf("Enter characters to be stuffed:");
	scanf("%s", a);
	printf("\nEnter a character that represents starting delimiter:");
	scanf(" %c", &sd);
	printf("\nEnter a character that represents ending delimiter:");
	scanf(" %c", &ed);
	x[0] = s[0] = s[1] = sd;
	x[1] = s[2] = '\0';
	y[0] = d[0] = d[1] = ed;
	d[2] = y[1] = '\0';
	strcat(fs, x);
	for(i = 0; i < strlen(a); i++)
```

```
        {
            t[0] = a[i];
            t[1] = '\0';
            if(t[0] == sd)
                strcat(fs, s);
            else if(t[0] == ed)
                strcat(fs, d);
            else
                strcat(fs, t);
        }
        strcat(fs, y);
        printf("\n After stuffing:%s", fs);
        getch();
    }
```

Output:-

Enter characters to be stuffed: goodday

Enter a character that represents starting delimiter: d

Enter a character that represents ending delimiter: g

After stuffing: dggooddddayg.

STUDY OF NETWORK DEVICES IN DETAIL

Aim: Study of following Network Devices in Detail

•Repeater

• Hub

• Switch

• Bridge

• Router • Gate Way

Apparatus (Software): No software or hardware needed.

Procedure: Following should be done to understand this practical.

1. Repeater: Functioning at Physical Layer.Arepeater is an electronic device that receives a signal and retransmits it at a higher level and/or higher power, or onto the other side of an obstruction, so that the signal can cover longer distances. Repeater have two ports ,so cannot be use to connect for more than two devices

2. Hub: An Ethernet hub, active hub, network hub, repeater hub, hub or concentrator is a device for connecting multiple twisted pair or fiber optic Ethernet devices together and making them act as a single network segment. Hubs work at the physical layer (layer 1) of the OSI model. The device is a form of multiport repeater. Repeater hubs also participate in collision detection, forwarding a jam signal to all ports if it detects a collision.

3. Switch: Anetwork switch or switching hub is a computer networking device that connects network segments.The term commonly refers to a network bridge that processes and routes data at the data link layer (layer 2) of the OSI model. Switches that additionally process data at the network layer (layer 3 and above) are often referred to as Layer 3 switches or multilayer switches.

4. Bridge: A network bridge connects multiple network segments at the data link layer (Layer 2) of the OSI model. In Ethernet networks, the term bridge formally means a device that behaves according to the IEEE 802.1D standard. A bridge and switch are very much alike; a switch being a bridge with numerous ports. Switch or Layer 2 switch is often used interchangeably with bridge .Bridges can analyze incoming data packets to determine if the bridge is able to send the given packet to another segment of the network.

5. Router: A router is an electronic device that interconnects two or more computer networks, and selectively interchanges packets of data between them. Each data packet contains address information that a router can use to determine if the source and destination are on the same network, or if the data packet must be transferred from one network to another. Where multiple routers are used in a large collection of interconnected networks, the routers exchange information about target system addresses, so that each router can build up a table showing the preferred paths between any two systems on the interconnected networks.

6. Gate Way: In a communications network, a network node equipped for interfacing with another network that uses different protocols. • A gateway may contain devices such as protocol translators, impedance matching devices, rate converters, fault isolators, or signal translators as necessary to provide system interoperability. It also requires the establishment of mutually acceptable administrative procedures between both networks. • A protocol translation/mapping

gateway interconnects networks with different network protocol technologies by performing the required protocol conversions.

IMPLEMENT CRC-CCITT POLYNOMIAL TO OBTAIN CRC CODE.

Aim:

For the given data, use CRC-CCITT polynomial to obtain CRC code. Verify the program for the cases. i) Without error. ii) With error.

Theory:

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 corrective action can be taken against presumed data corruption if the check values do not match. Example: To compute an n-bit binary CRC, line the bits representing the input in a row, and position the (n+1)-bit pattern representing the CRC's divisor (called a "polynomial") underneath the left-hand end of the row. Start with the message to be encoded: 11010011101100 This is first padded with zeroes corresponding to the bit length n of the CRC.

Here is the first calculation for computing a 3-bit CRC: 11010011101100 000 INPUT RIGHT PADDED WITH ZERO BITS 1011 DIVISOR (4 BITS)

---------- 01100011101100 000 RESULT If the input bit above the leftmost divisor bit is 0, do nothing. If the input bit above the leftmost divisor bit is 1, the divisor is XORed into the input. The divisor is then shifted one bit to the right, and the process is repeated until the divisor reaches the right-hand end of the input row. Here is the entire calculation:

Program:

```
//crc can detect all single bit error, double bit, odd bits of error and burst error

#include<stdio.h>

#include<string.h>

#define N strlen(g)

char t[28],cs[28],g[]="10001000000100001";

int a,i,j;

void xor()

{

for(j = 1;j < N; j++)

cs[j] = (( cs[j] == g[j])?'0':'1');

}

void crc()

{

for(i=0;i<N;i++)

 cs[i]=t[i];
```

```
 do
{
 if(cs[0]=='1')
 xor();
 for(j=0;j<N-1;j++)
 cs[j]=cs[j+1];
 cs[j]=t[i++];
 }while(i<=a+N-1);
}
int main()
{
 printf("\nEnter data : ");
 scanf("%s",t);
 printf("\n---------------------------------------");
 printf("\nGeneratng polynomial : %s",g);
 a=strlen(t);
 for(i=a;i<a+N-1;i++)
 t[i]='0';
 printf("\n---------------------------------------");
 printf("\nModified data is : %s",t);
 printf("\n---------------------------------------");
 crc();
 printf("\nChecksum is : %s",cs);
 for(i=a;i<a+N-1;i++)
 t[i]=cs[i-a];
 printf("\n---------------------------------------");
 printf("\nFinal codeword is : %s",t);
 printf("\n---------------------------------------");
 printf("\nEnter received message ");
 scanf("%s",t);
 crc();
```

```
 for(i=0;(i<N-1) && (cs[i]!='1');i++);
 if(i<N-1)
 printf("\nError detected\n\n");
 else
 printf("\nNo error detected\n\n");
 printf("\n----------------------------------------\n");
 return 0;
}
```

IMPLEMENT STOP AND WAIT PROTOCOL AND SLIDING WINDOW PROTOCOL

Aim: Implement Stop and Wait Protocol and Sliding Window Protocol in a C program and execute the same and display the result.

(i) Stop and Wait Protocol

Theory:

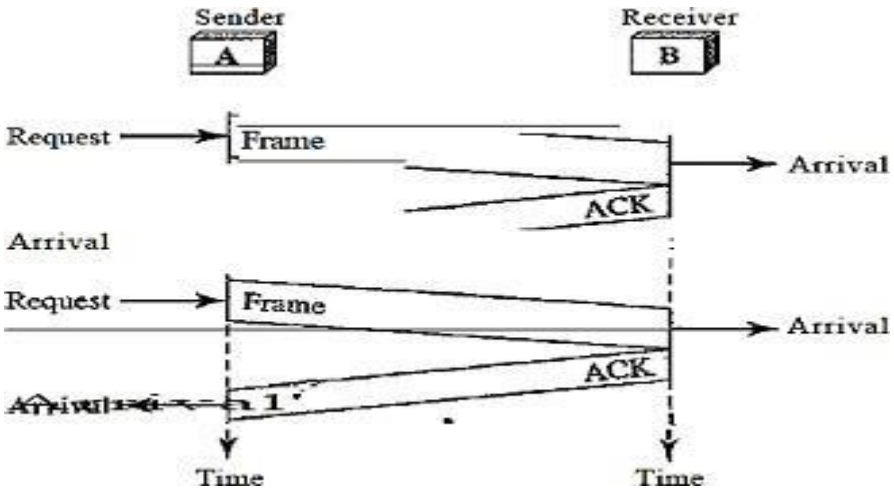
If data frames arrive at the receiver site faster than they can be processed, the frames must be stored until their use. Normally, the receiver does not have enough storage space, especially if it is receiving data from many sources.

This may result in either the discarding of frames or denial of service. To prevent the receiver from becoming overwhelmed with frames, we somehow need to tell the sender to slow down. There must be feedback from the receiver to the sender.

The protocol we discuss now is called the Stop-and-Wait Protocol because the sender sends one frame, stops until it receives confirmation from the receiver (okay to go ahead), and then sends the next frame. We still have unidirectional communication for data frames, but auxiliary ACK frames (simple tokens of acknowledgment) travel from the other direction. We add flow control to our previous protocol.

Example:

Figure shows an example of communication using this protocol. It is still very simple. The sender sends one frame and waits for feedback from the receiver.



When the ACK arrives, the sender sends the next frame. Note that sending two frames in the protocol involves the sender in four events and the receiver in two events.

Program:

```
#include
<stdio.h>
#include
<stdlib.h>
#define
RTT 4
#define TIMEOUT 4
#define
```

```
TOT_FRAMES 7
enum {NO,YES}
ACK;
int main()
{
int wait_time,i=1;
ACK=YES;
for(;i<=TOT_FRA
MES;)
{
if (ACK==YES && i!=1)
{
printf("\nSENDER: ACK for Frame %d Received.\n",i-1);
}
printf("\nSENDER: Frame %d sent, Waiting for
ACK...\n",i); ACK=NO;
wait_time= rand()
% 4+1; if
(wait_time==TIME
OUT)
{
printf("SENDER: ACK not received for Frame %d=>TIMEOUT Resending Frame...",i);
}
else
{
sleep(RTT);
printf("\nRECEIVER: Frame %d received, ACK
sent\n",i); printf("..........."); ACK=YES;
i++;
}
}
return 0;
}
```

(ii) Sliding Window Protocol :

Theory:

Sliding Window:

In this protocol (and the next), the sliding window is an abstract concept that defines the range of sequence numbers that is the concern of the sender and receiver. In other words, the sender and receiver need to deal with only part of the possible sequence numbers. The range which is the concern of the sender is called the send sliding window; the range that is the concern of the receiver is called the receive sliding window. We discuss both here. The send window is an imaginary box covering the sequence numbers of the data frames which can be in transit. In each window position, some of these sequence numbers define the frames that have been sent; others define those that can be sent. The maximum size of the window is 2m-1 for reasons that we discuss later. In this chapter, we let the size be fixed and set to the maximum value, but we will see in future chapters that some protocols may have a variable window size

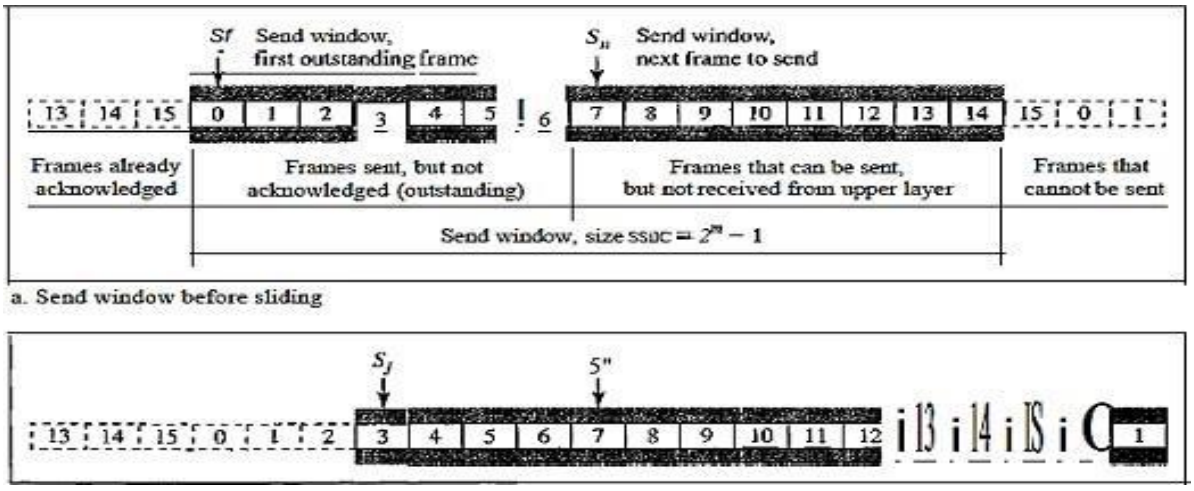
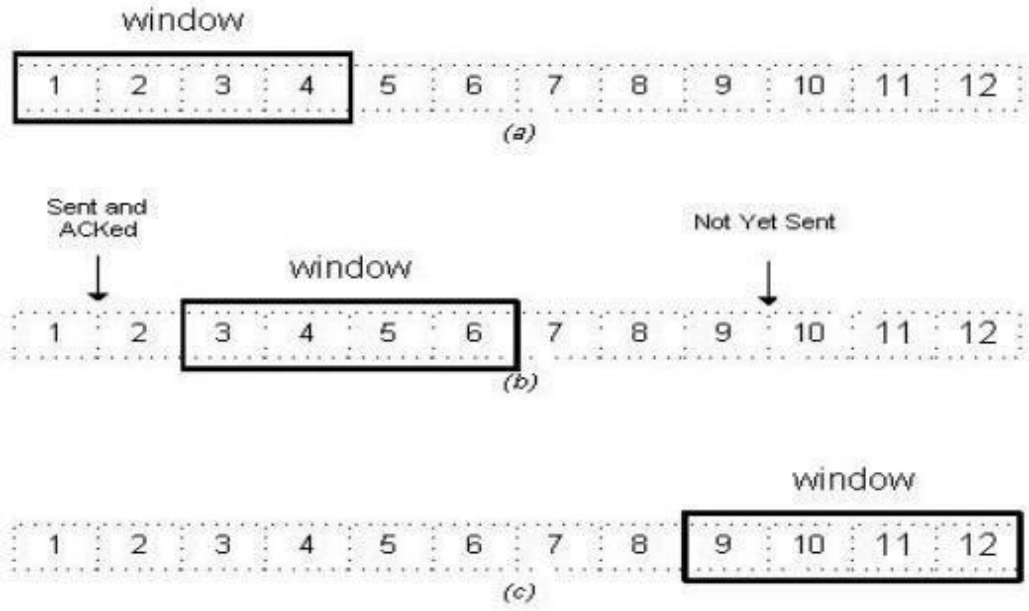


Figure shows a sliding window of size 15 (m =4). The window at any time divides the possible sequence numbers into four regions.

The first region, from the far left to the left wall of the window, defines the sequence numbers belonging to frames that are already acknowledged. The sender does not worry about these frames and keeps no copies of them. The second region, colored in Figure defines the range of sequence numbers belonging to the frames that are sent and have an unknown status. The sender needs to wait to find out if these frames have been received or were lost. We call these outstanding frames.

The third range, white in the figure, defines the range of sequence numbers for frames that can be sent; however, the corresponding data packets have not yet been received from the network layer. Finally, the fourth region defines



sequence numbers that cannot be used until the window slides.

In Networking, Window simply means a buffer which has data frames that needs to be transmitted. Both sender and receiver agrees 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.

Program:

```
#include
<stdio.h>
#include
<stdlib.h>
#define
RTT 5
int main()
{
                int
                window_siz
                e,i,f,frames[
                50];
                printf("Enter
                window
                size: ");
                scanf("%d",
                &window_si
                ze);
                printf("\nEnter number of
                frames to transmit: ");
                scanf("%d",&f);
                printf("\nEnter %d frames: ",f);

               for(i=
               1;i<=f
               ;i++)
               scanf(
               "%d",
               &fra
               mes[i]
               );
               printf("\nAfter sending %d frames at each stage sender waits for
               ACK",window_size);
               printf("\nSending frames in the following manner... \n\n");

for(i=1;i<=f;i++)
                {
                  if(i%window_size!=0)
                {
                   printf("%d",frames[i]);
                }
                  else
                {
                   printf("
                   %d\n",frames[i]);
                   printf("SENDER:
```

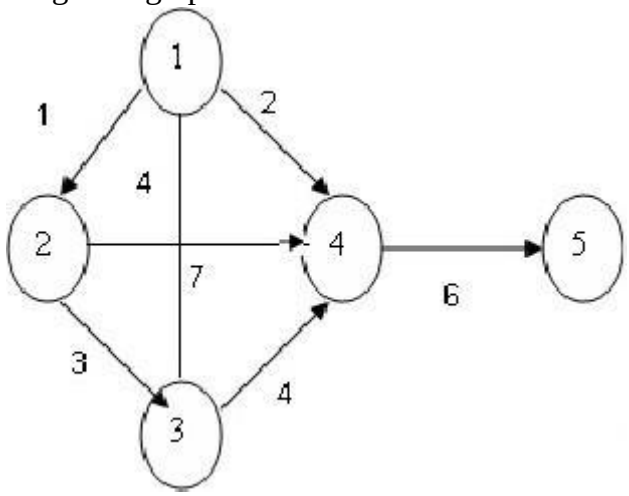
```
    waiting for ACK...\n\n");
    sleep(RTT/2);
    printf("RECEIVER: Frames
    Received, ACK Sent\n"); printf("..........\n"); sleep(RTT/2);
    printf("SENDER:ACK received, sending next frames\n");
}
}

if(f%window_size!=0)
{
  printf("\nSENDER: waiting
  for ACK...\n"); sleep(RTT/2);
printf("\nRECEIVER:Frames
Received, ACK Sent\n"); printf("..........\n"); sleep(RTT/2);
printf("SENDER:ACK received.");
}
return 0;
}
```

Result:

Both the protocols are implemented and executed and the result is displayed on the screen.

IMPLEMENT DIJKSTRA‘S ALGORITHM TO COMPUTE THE SHORTEST ROUTING PATH

OBJECTIVE: Implement Dijkstra‘s algorithm to compute the Shortest path thru agiven graph.



RESOURCE: Turbo C

Program Logic: Dijkstra's algorithm is very similar to Prim's algorithm for minimum spanning tree. Like Prim's MST, we generate a SPT (shortest path tree) with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, and other set includes vertices not yet included in shortest path tree.
PROCEDURE: Go to debug -> run or press CTRL + F9 to run the program.

SOURCE CODE:

```
// .PROGRAM FOR FINDING SHORTEST PATH FOR A GIVEN GRAPH//

#include<stdio.
h>
#include<conio
.h>void main()

{

int path[5][5],i,j,min,a[5][5],p,st=1,ed=5,stp,edp,t[5],index;
clrscr();

printf("enter the cost
matrix\n");for(i=1;i<=5;i++)
for(j=1;j<=5;j++)
scanf("%d",&a[i][j]);
printf("enter the paths\n");
scanf("%d",&p);

printf("enter possible
paths\n");for(i=1;i<=p;i++)
```

```
for(j=1;j<=5;j++)
scanf("%d",&path[i][j]);

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

{
t
[
i
]
=
0
;

stp=st;
for(j=1;j<=5;j
++)

{

edp=path[i][j+1];
t[i]=t[i]+a[stp][ed
p];if(edp==ed)

bre
ak;
els
e
stp
=ed
p;

}

}min=t[st];
index=st;
for(i=1;i<=p;i
++)

{

if(min>t[i])

{

min
=t[i
];
inde
x=i;

}
```

```
}

printf("minimum cost
%d",min); printf("\n minimum
cost path ");for(i=1;i<=5;i++)

{

printf("-->
%d",path[index][i]);
if(path[index][i]==ed)

break;

}

getch();

}
```

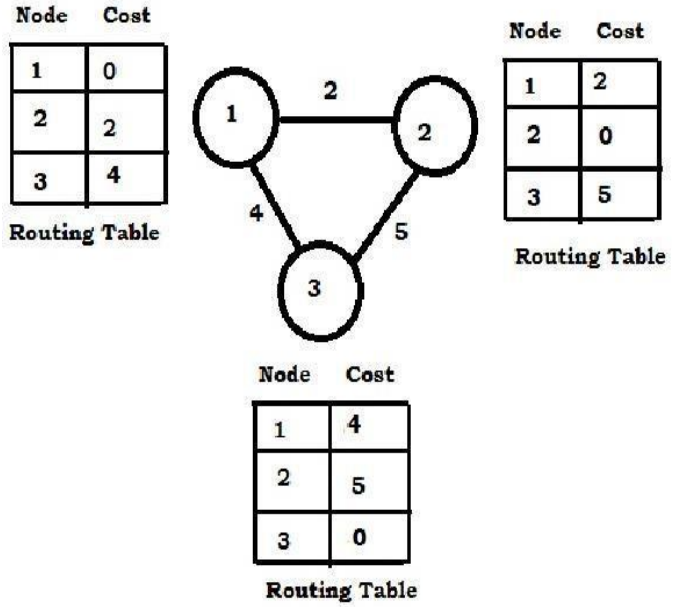
Output:

Viva questions:

1. Define Dijkstra's algorithm?
2. What is the use of Dijkstra's algorithm?
3. What is path?
4. What is minimum cost path?
5. How to find shortest path using Dijkstra's algorithm?

Distance Vector Routing Algorithm using Bellman Ford's Algorithm

OBJECTIVE: Obtain Routing table at each node using distance vector routing algorithm for agiven subnet.



PROGRAM LOGIC:

Distance Vector Routing Algorithms calculate a best route to reach a destination based solely on distance. E.g. RIP. RIP calculates the reach ability based on hop count. It' s different from link state algorithms which consider some other factors like bandwidth and other metrics to reach a destination. Distance vector routing algorithms are not preferable for complex networks and take longer to converge.

PROCEDURE: Go to debug -> run or press CTRL + F9 to run the program.

SOURCE CODE:

```
#include<stdio.h>

struct node

{

   unsigned dist[20];

   unsigned from[20];

}rt[10];

int main()

{

   int costmat[20][20];

   int nodes,i,j,k,count=0;
```

```
printf("\nEnter the number of nodes : ");
scanf("%d",&nodes);//Enter the nodes
printf("\nEnter the cost matrix :\n");
for(i=0;i<nodes;i++)
{
   for(j=0;j<nodes;j++)
   {
      scanf("%d",&costmat[i][j]);
      costmat[i][i]=0;
      rt[i].dist[j]=costmat[i][j];//initialise the distance equal to cost matrix
      rt[i].from[j]=j;
   }
}
   do
   {
      count=0;
      for(i=0;i<nodes;i++)//We choose arbitary vertex k and we calculate the direct distance from the node i to k using the cost matrix
      //and add the distance from k to node j
      for(j=0;j<nodes;j++)
      for(k=0;k<nodes;k++)
         if(rt[i].dist[j]>costmat[i][k]+rt[k].dist[j])
         {//We calculate the minimum distance
            rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];
            rt[i].from[j]=k;
            count++;
         }
   }while(count!=0);
   for(i=0;i<nodes;i++)
```

```
    {
        printf("\n\n For router %d\n",i+1);
        for(j=0;j<nodes;j++)
        {
            printf("\t\nnode %d via %d Distance %d ",j+1,rt[i].from[j]+1,rt[i].dist[j]);
        }
    }
  printf("\n\n");
  getch();
}
```

Output:

```
Enter the number of nodes :
3
Enter the cost matrix :
0 2 7
2 0 1
7 1 0
For router 1
node 1 via 1 Distance 0
node 2 via 2 Distance 2
node 3 via 3 Distance 3
For router 2
node 1 via 1 Distance 2
node 2 via 2 Distance 0
node 3 via 3 Distance 1
For router 3
node 1 via 1 Distance 3
node 2 via 2 Distance 1
node 3 via 3 Distance 0
```

Viva Questions:

1. What is routing?
2. What is best algorithm among all routing algorithms?
3. What is static routing?
4. Differences between static and dynamic?
5. What is optimality principle?

SOCKET PROGRAM FOR ECHO.

AIM

To write a socket program for implementation of echo.

ALGORITHM

CLIENT SIDE

1. Start the program.
2. Create a socket which binds the Ip address of server and the port address to acquire service.
3. After establishing connection send a data to server.
4. Receive and print the same data from server.
5. Close the socket.
6. End the program.

SERVER SIDE

1. Start the program.
2. Create a server socket to activate the port address.
3. Create a socket for the server socket which accepts the connection.
4. After establishing connection receive the data from client.
5. Print and send the same data to client.
6. Close the socket.
7. End the program.

PROGRAM

ECHO CLIENT

```
import java.io.*;
import java.net.*;
public class eclient
{
public static void main(String args[])
{
Socket c=null;
String line;
DataInputStream is,is1;
PrintStream os;
try
{
c=new Socket("localhost",8080);
}
catch(IOException e)
{
System.out.println(e);
}
try
{
```

```
os=new PrintStream(c.getOutputStream());
is=new DataInputStream(System.in);
is1=new DataInputStream(c.getInputStream());
do
{
System.out.println("client");
line=is.readLine();
os.println(line);
if(!line.equals("exit"))
System.out.println("server:"+is1.readLine());
}while(!line.equals("exit"));
}
catch(IOException e)
{
System.out.println("socket closed");
}
}
}
```

Echo Server:

```
import java.io.*;
import java.net.*;
import java.lang.*;
public class eserver
{
public static void main(String args[])throws IOException
{
ServerSocket s=null;
String line;
DataInputStream is;
PrintStream ps;
Socket c=null;
try
{
s=new ServerSocket(8080);
}
catch(IOException e)
{
System.out.println(e);
}
try
{
c=s.accept();
is=new DataInputStream(c.getInputStream());
ps=new PrintStream(c.getOutputStream());
while(true)
{
line=is.readLine();
System.out.println("msg received and sent back to client");
ps.println(line);
```

```
}
}
catch(IOException e)
{
System.out.println(e);
}
}
}
```

OUTPUT

CLIEN

Enter the IP address 127.0.0.1

CONNECTION ESTABLISHED

Enter the data: **HELLO**

Client received: **HELLO**

SERVER

CONNECTION ACCEPTED

Server received: **HELLO**

RESULT

Thus the program for simulation of echo server was written & execute

CLIENT- SERVER APPLICATION FOR CHAT

AIM

To write a client-server application for chat using TCP

ALGORITHM

CLIENT

1. Start the program
2. Include necessary package in java
3. To create a socket in client to server.
4. The client establishes a connection to the server.
5. The client accept the connection and to send the data from client to server.
6. The client communicates the server to send the end of the message
7. Stop the program.

SERVER

1. Start the program
2. Include necessary package in java
3. To create a socket in server to client
4. The server establishes a connection to the client.
5. The server accept the connection and to send the data from server to client and
6. vice versa
7. The server communicate the client to send the end of the message.
8. Stop the program.

PROGRAM

TCPserver1.java

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

public class TCPserver1
{
 public static void main(String arg[])
{
ServerSocket s=null;
String line;
DataInputStream is=null,is1=null;
PrintStream os=null;
Socket c=null;
try
{
s=new ServerSocket(9999);
catch(IOException e)
{
System.out.println(e);
 }
```

```
try
{
c=s.accept();
is=new DataInputStream(c.getInputStream());
is1=new DataInputStream(System.in);
os=new PrintStream(c.getOutputStream());
do
{
line=is.readLine();
System.out.println("Client:"+line);
System.out.println("Server:");
line=is1.readLine();
os.println(line);
}
while(line.equalsIgnoreCase("quit")==false);
is.close();
os.close();
}
catch(IOException e)
{
System.out.println(e);
}
}
}
```

TCPclient1.java

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

 public class TCPclient1
{
 public static void main(String arg[])
{
Socket c=null;
String line;
DataInputStream is,is1;
PrintStream os;
try
{
c=new Socket("10.0.200.36",9999);
}
```

```
catch(IOException e)
{
System.out.println(e);
}
try
{
 os=new PrintStream(c.getOutputStream());
is=new DataInputStream(System.in);
is1=new DataInputStream(c.getInputStream());
do
{
System.out.println("Client:");
line=is.readLine();
os.println(line);
System.out.println("Server:" + is1.readLine());
}
while(line.equalsIgnoreCase("quit")==false);
is1.close();
os.close();
}
catch(IOException e)
{
System.out.println("Socket Closed!Message Passing is over");
}}
```

OUTPUT :

SERVER

```
 C:\Program Files\Java\jdk1.5.0\bin>javac TCPserver1.java
 Note: TCPserver1.java uses or overrides a deprecated API.
 Note: Recompile with -deprecation for details.
C:\Program Files\Java\jdk1.5.0\bin>java TCPserver1

Client: Hai Server
Server:Hai Client
Client: How are you
Server:Fine
Client: quit
Server:quit
```

CLIENT

```
C:\Program Files\Java\jdk1.5.0\bin>javac TCPclient1.java
 Note: TCPclient1.java uses or overrides a deprecated API.
Note: Recompile with -deprecation for details.
C:\Program Files\Java\jdk1.5.0\bin>java TCPclient1

Client:Hai Server
Server: Hai Client
Client:How are you
Server: Fine
Client:quit
Server: quit
```

RESULT

Thus the above program a client-server application for chat using TCP / IP was executed and successfully.

CASE STUDY ON ROUTING ALGORITHMS

AIM:

To study the various routing algorithms

DESCRIPTION

1. Link state routing algorithm
2. Flooding
3. Distance vector routing algorithm

LINK STATE ROUTING

AIM

To study the link state routing

LINK STATE ROUTING

Routing is the process of selecting best paths in a network. In the past, the term routing was also used to mean forwarding network traffic among networks. However this latter function is much better described as simply forwarding. Routing is performed for many kinds of networks, including the telephone network (circuit switching), electronic data networks (such as the Internet), and transportation networks. This article is concerned primarily with routing in electronic data networks using packet switching technology.

In packet switching networks, routing directs packet forwarding (the transit of logically addressed network packets from their source toward their ultimate destination) through intermediate nodes. Intermediate nodes are typically network hardware devices such as routers, bridges, gateways, firewalls, or switches. General-purpose computers can also forward packets and perform routing, though they are not specialized hardware and may suffer from limited performance. The routing process usually directs forwarding on the basis of routing tables which maintain a record of the routes to various network destinations. Thus, constructing routing tables, which are held in the router's memory, is very important for efficient routing. Most routing algorithms use only one network path at a time. Multipath routing techniques enable the use of multiple alternative paths.

In case of overlapping/equal routes, the following elements are considered in order to decide which routes get installed into the routing table (sorted by priority):

1. Prefix-Length: where longer subnet masks are preferred (independent of whether it is within a routing protocol or over different routing protocol)
2. Metric: where a lower metric/cost is preferred (only valid within one and the same routing protocol)
3. Administrative distance: where a lower distance is preferred (only valid between different routing protocols)

Routing, in a more narrow sense of the term, is often contrasted with bridging in its assumption that network addresses are structured and that similar addresses imply proximity within the network. Structured addresses allow a single routing table entry to represent the route to a group of devices. In large networks, structured addressing (routing, in the narrow sense) outperforms unstructured addressing (bridging). Routing has become the dominant form of addressing on the Internet. Bridging is still widely used within localized environments.

2. FLOODING

Flooding s a simple routing algorithm in which every incoming packet is sent through every outgoing link except the one it arrived on.Flooding is used in bridging and in systems such as Usenet and peer-to-peer file sharing and as part of some routing protocols, including OSPF, DVMRP, and those used in ad-hoc wireless networks.There are generally two types of flooding available, Uncontrolled Flooding and Controlled Flooding.Uncontrolled Flooding is the fatal law of flooding. All nodes have neighbours and route packets indefinitely. More than two neighbours creates a broadcast storm.

Controlled Flooding has its own two algorithms to make it reliable, SNCF (Sequence Number Controlled Flooding) and RPF (Reverse Path Flooding). In SNCF, the node attaches its own address and sequence number to the packet, since every node has a memory of addresses and sequence numbers. If it receives a packet in memory, it drops it immediately while in RPF, the node will only send the packet forward. If it is received from the next node, it sends it back to the sender.

ALGORITHM

There are several variants of flooding algorithm. Most work roughly as follows:

1. Each node acts as both a transmitter and a receiver.

2. Each node tries to forward every message to every one of its neighbours except the source node.

This results in every message eventually being delivered to all reachable parts of the network.

Algorithms may need to be more complex than this, since, in some case, precautions have to be taken to avoid wasted duplicate deliveries and infinite loops, and to allow messages to eventually expire from the system. A variant of flooding called selective flooding partially addresses these issues by only sending packets to routers in the same direction. In selective flooding the routers don't send every incoming packet on every line but only on those lines which are going approximately in the right direction.

3. DISTANCE VECTOR

In computer communication theory relating to packet-switched networks, a distance- vector routing protocol is one of the two major classes of routing protocols, the other major class being the link-state protocol. Distance-vector routing protocols use the Bellman–Ford algorithm, Ford–Fulkerson algorithm, or DUAL FSM (in the case of Cisco Systems's protocols) to calculate paths.

A distance-vector routing protocol requires that a router informs its neighbors of topology changes periodically. Compared to link-state protocols, which require a router to inform all the nodes in a network of topology changes, distance-vector routing protocols have less computational complexity and message overhead.

The term distance vector refers to the fact that the protocol manipulates vectors (arrays) of distances to other nodes in the network. The vector distance algorithm was the original ARPANET routing algorithm and was also used in the internet under the name of RIP (Routing Information Protocol).

Examples of distance-vector routing protocols include RIPv1 and RIPv2 and IGRP. Method

Routers using distance-vector protocol do not have knowledge of the entire path to a destination. Instead they use two methods:

1. Direction in which router or exit interface a packet should be forwarded.

2. Distance from its destination

Distance-vector protocols are based on calculating the direction and distance to any link in a network. "Direction" usually means the next hop address and the exit interface. "Distance" is a measure of the cost to reach a certain node. The least cost route between any two nodes is the route with minimum distance. Each node maintains a vector (table) of minimum distance to every node. The cost of reaching a destination is calculated using various route metrics. RIP uses the hop count of the destination whereas IGRP takes into account other information such as node delay and available bandwidth.

Updates are performed periodically in a distance-vector protocol where all or part of a router's routing table is sent to all its neighbors that are configured to use the same distance- vector routing protocol. RIP supports cross-platform distance vector routing whereas IGRP is a Cisco Systems proprietary distance vector routing protocol. Once a router has this information it is able to amend its own routing table to reflect the changes and then inform its neighbors of the changes. This process has been described as ‗routing by rumor' because routers are relying on the information they receive from other routers and cannot determine if the information is actually valid and true. There are a number of features which can be used to help with instability and inaccurate routing information.

EGP and BGP are not pure distance-vector routing protocols because a distance-vector protocol calculates routes based only on link costs whereas in BGP, for example, the local route preference value takes priority over the link cost.

Count-to-infinity problem

The Bellman–Ford algorithm does not prevent routing loops from happening and suffers from the count-to-infinity problem. The core of the count-to-infinity problem is that if A tells B that it has a path somewhere, there is no way for B to know if the path has B as a part of it. To see the problem clearly, imagine a subnet connected like A–B–C–D–E–F, and let the metric between the routers be "number of jumps". Now suppose that A is taken offline. In the vector-update-process B notices that the route to A, which was distance 1, is down – B does not receive the vector update from A. The problem is, B also gets an update from C, and C is still not aware of the fact that A is down – so it tells B that A is only two jumps from C (C to B to A), which is false. This slowly propagates through the network until it reaches infinity (in which case the algorithm corrects itself, due to the relaxation property of Bellman–Ford).

RESULT

Thus the study about the various routing algorithms has been completed successfully.