DELHI TECHNOLOGICAL UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

COMPUTER NETWORKS (CO-306)

LAB FILE

Submitted to:

Mr. Sanjay Kumar
Assistant Professor
Department of Computer Science
Delhi Technological University

Submitted by:

Kunal Sinha
2K17/CO/164
Computer Science(A3)

VISION

Department of Computer Science & Engineering to be a leading world class technology department playing its role as a key node in national and global knowledge network, thus empowering the computer science industry with the wings of knowledge and power of innovation.

MISSION

1. To nurture talent of students for research, innovation and excellence in the field of computer engineering starting from Under graduate level.
2. To develop highly analytical and qualified computer engineers by imparting training on cutting edge technology.
3. To produce socially sensitive computer engineers with professional ethics.
4. To focus on R&D environment in close partnership with industry and foreign universities.
5. To produce well-rounded, up to date, scientifically tempered, design oriented engineer and scientists capable of lifelong learning.

Programme Educational Objectives (PEOs)

---

1. To acquire in-depth knowledge of software and hardware techniques, which provide a strong foundation to pursue continuing education and nurture the talent for innovation and research.
2. To nurture the talent in leadership qualities, at an appropriate level in order to address the issues in a responsive, ethical and innovative manner.
3. To excel in careers by being a part of success and growth of an organization with whom they will be associated.
4. To inculcate the ability for lifelong learning by active participation in self-study courses, seminars, research projects.

Programme Specific Outcomes (PSOs)

---

1. Design, analyze and develop the engineering problems.
2. Specify, design, develop, test and maintain usable systems that behave reliably and efficiently.
3. Develop systems that would perform tasks related to Research, Education and Training and/or E-governance.

INDEX

S.No	Experiment	Date	Signature	Remarks
1	Write a program to implement Framing Techniques: Bit Stuffing, Byte Stuffing and Character Stuffing	07-01-2020		
2	Write a program to implement Cyclic Redundancy Check (CRC)	14-01-2020		
3	Write a program to implement stop and wait protocol.	21-01-2020		
4	Write a program to implement sliding window protocol.	28-01-2020		
5	To display the Network Class, Network ID and Host ID of a given IP Address	04-02-2020		
6	Write a program to implement Distance vector routing algorithm.	11-02-2020		
7	Write a program to implement Link state routing algorithm	18-02-2020		

Experiment 1

- **OBJECTIVE** : Write a program to implement Framing Techniques: Bit Stuffing, Byte Stuffing and Character Stuffing.

- **THEORY** : Bit stuffing is the process of inserting non-information bits into data to break up bit patterns to affect the synchronous transmission of information. It is widely used in network and communication protocols, in which bit stuffing is a required part of the transmission process. Bit stuffing is commonly used to bring bit streams up to a common transmission rate or to fill frames. Bit stuffing is also used for run-length limited coding.

  Byte stuffing is a process that transforms a sequence of data bytes that may contain 'illegal' or 'reserved' values (such as packet delimiter) into a potentially longer sequence that contains no occurrences of those values. The extra length of the transformed sequence is typically referred to as the overhead of the algorithm. The COBS algorithm tightly bounds the worst-case overhead, limiting it to a minimum of one byte and a maximum of ⌈n/254⌉ bytes (one byte in 254, rounded up). Consequently, the time to transmit the encoded byte sequence is highly predictable, which makes COBS useful for real-time applications in which jitter may be problematic. The algorithm is computationally inexpensive and its average overhead is low compared to other unambiguous framing algorithms.

  In Character Stuffing, each frame starts with the ASCII character sequence DLE STX and ends with the sequence DLE ETX. This method overcomes the drawbacks of the character count method. If the destination ever loses synchronization, it only has to look for DLE STX and DLE ETX characters. If however, binary data is being transmitted then there exists a possibility of the characters DLE STX and DLE ETX occurring in the data. Since this can interfere with the framing, a technique called character stuffing is used. The sender's data link layer inserts an ASCII DLE character just before the DLE character in the data. The receiver's data link layer removes this DLE before this data is given to the network layer. However character stuffing is closely associated with 8-bit characters and this is a major hurdle in transmitting arbitrary sized characters The project is specifically designed for the use of librarians and library users. The product will work as a complete user interface for library management process and library usage from ordinary users. Library management system can he used by an existing or new library to manage its books and book borrowing, insertion and monitoring. It is especially useful for any educational institute where modifications in the content can be done easily according to requirements.

- **ALGORITHM :**
  Bit stuffing

  1. Start
  2. Initialize the array for transmitted stream with the special bit pattern 0 1 1 1 1 1 1 0, which indicates the beginning of the frame.
  3. Get the bit stream to be transmitted in the array.
  4. Check for 5 consecutive ones and if they occur, stuff a bit 0.
  5. Display the data transmitted on the data line after appending 0 1 1 1 1 1 1 0 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. End

Byte stuffing

1. Start
2. Append 0 1 1 1 1 1 1 0 at the beginning of the string.
3. Check the data if character is present.
   If character is present in the string, insert another 0 1 1 1 1 1 1 0 in the string.
4. Transmit 0 1 1 1 1 1 1 0 at the end of the string.
5. Display the string.
6. End

Character stuffing

1. Start
3. Check the data if character is present. If character DLE is present in the string, insert another DLE in the string.
4. Transmit DLE STX at the end of the string
5. Display the string
6. End

- **CODE:**

Bit stuffing

```
#include<iostream>
using namespace std;
int main()
{
int a[50];
int i, j, k, n, c, pos;
c = 0;
pos = 0;
cout<<"Enter number of bits:\n";
cin>>n;
cout<<"Enter the bits:\n";
for(i = 0; i < n; i++)
cin>>a[i];
for(i = 0; i < n; i++)
{
if(a[i] == 1)
{
c++;
if(c == 5)
{
pos = i+1;
c = 0;
for(j = n-1; j > pos; j--)
{
```

```
k = j+1;
a[k] = a[j];
}
a[pos] = 0;
n++;
}
}
else
c = 0;
}
cout<<"Data after stuffing:\n";
cout<<"01111110";
for(i = 0; i < n; i++)
{
cout<<a[i];
}
cout<<"01111110"<<endl;
return 0;
}
```

Byte stuffing

```
#include<iostream>
using namespace std;
int main()
{
int i,j,k,data[100],n;
cout<<"Enter the no. of bits: "<<endl;
cin>>n;
cout<<"Enter the bits to be transmitted: "<<endl;
for(i=0;i<n;i++)
{
cin>>data[i];
}
cout<<endl;
cout<<"The bits entered are: "<<endl;
for(i=0;i<n;i++)
{
cout<<data[i];
}
cout<<endl;
cout<<"The byte stuffed array is: "<<endl;
cout<<"01111110 ";
for(i=0;i<n;i++)
{
```

```
if(data[i]==0 && data[i+1]==1 && data[i+2]==1 && data[i+3]==1 && data[i+4]==1 &&
data[i+5]
&& data[i+6]==1 && data[i+7]==0)
{
cout<<"01111110011111110";
i=i+7;
}
else cout<<data[i];
}
cout<<" 01111110"<<endl;
return 0;
}
```

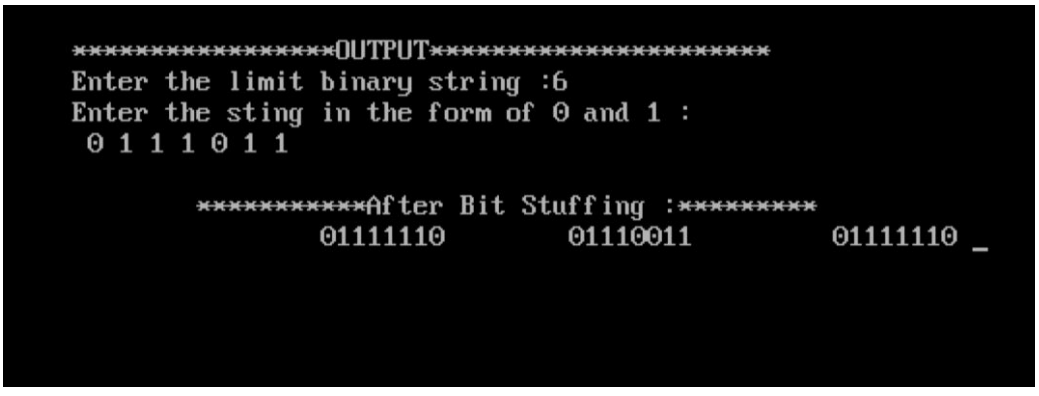
Character stuffing

```
#include <iostream>
using namespace std;
int main()
{
char c[50],d[50],t[50];
int i,m,j;
cout << "Enter the number of characters : ";
cin >> m;
cout << "\nEnter the characters (without spaces) : ";
for(i=0;i<m;i++)
cin >> c[i];
printf("\nOriginal data : ");
for(i=0;i<m;i++)
cout << c[i];
d[0]='d';
d[1]='l';
d[2]='e';
d[3]='s';
d[4]='t';
d[5]='x';
for(i=0,j=6;i<m;i++,j++)
{
if((c[i]=='d'&&c[i+1]=='l'&& c[i+2]=='e'))
{
d[j]='d';
j++;
d[j]='l';
j++;
d[j]='e';
```

```
j++;
m=m+3;
}
d[j]=c[i];
}
m=m+6; m++;
d[m]='d'; m++;
d[m]='l'; m++;
d[m]='e' m++;
d[m]='s'; m++;
d[m]='t'; m++;
d[m]='x'; m++;
cout << "\n\nTransmitted data: ";
for(i=0;i<m;i++)
cout << d[i];
for(i=6,j=0;i<m-6;i++,j++)
{
if(d[i]=='d'&&d[i+1]=='l'&&d[i+2]=='e'&&d[i+3]=='d'&&d[i+4]=='l'&&d[i+5]=='e')
i=i+3;
t[j]=d[i];
}
cout << "\nReceived data:";
for(i=0;i<j;i++)
{
cout << t[i];
}
}
```

- **OUTPUT:**

Bit Stuffing



Character Stuffing

```
Enter the number of characters : 10

Enter the characters (without spaces) : sopdlestxa

Original data : sopdlestxa

Transmitted data: dlestxsopdledlestxa◆dlestx
Received data:sopdlestxa

...Program finished with exit code 0
Press ENTER to exit console.
```

- **DISCUSSIONS:**
  The Data Link Layer is the second layer in the OSI model, above the Physical Layer, which ensures that the error free data is transferred between the adjacent nodes in the network. It breaks the datagrams passed down by above layers and convert them into frames ready for transfer. This is called Framing. Reliable data transfer service between two peer network layers. Flow Control mechanism which regulates the flow of frames such that data congestion is not there at slow receivers due to fast senders.

- **FINDING AND LEARNINGS:**
  Bit stuffing is the insertion of non-information bits into data. Note that stuffed bits should not be confused with overhead bits. Overhead bits are non-data bits that are necessary for transmission (usually as part of headers, checksums etc.).

  Byte Stuffing:A byte (usually escape character(ESC)), which has a predefined bit pattern is added to the data section of the frame when there is a character with the same pattern as the flag. Whenever the receiver encounters the ESC character, it removes from the data section and treats the next character as data, not a flag. But problem arises when text contains one or more escape characters followed by a flag. To solve this problem, the escape characters that are part of the text are marked by another escape character i.e., if the escape character is part of the text, an extra one is added to show that the second one is part of the text.

  Character Stuffing: 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.

Experiment 2

- **OBJECTIVE**: Write a program to implement Cyclic Redundancy Check (CRC)

- **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, in the event the check values do not match, corrective action can be taken against data corruption. CRCs can be used for error correction. CRCs are so called because the check (data verification) value is a redundancy (it expands the message without adding information) and the algorithm is based on cyclic codes. CRCs are popular because they are simple to implement in binary hardware, easy to analyse mathematically, and particularly good at detecting common errors caused by noise in transmission channels. Because the check value has a fixed length, the function that generates it is occasionally used as a hash function.

- **ALGORITHM:**
  1. A string of n as is appended to the data unit. The length of predetermined divisor is n+ 1.
  2. The newly formed data unit i.e. original data + string of n as are divided by the divisor using binary division and remainder is obtained. This remainder is called CRC.
  3. Now, string of n Os appended to data unit is replaced by the CRC remainder (which is also of n bit).
  4. The data unit + CRC is then transmitted to receiver.
  5. The receiver on receiving it divides data unit + CRC by the same divisor & checks the remainder.
  6. If the remainder of division is zero, receiver assumes that there is no error in data and it accepts it.
  7. If remainder is non-zero then there is an error in data and receiver rejects it.

- **CODE:**

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main()
{
int i, j, keylen, msglen;
char input[100], key[30], temp[30], quot[100], rem[30], key1[30];
printf("Enter Data: ");
gets(input);
printf("Enter Key: ");
gets(key);
keylen = strlen(key);
msglen = strlen(input);
strcpy(key1, key);
for(i = 0; i < keylen - 1; i++)
{
input[msglen+i] = '0';
```

```
}
for(i = 0; i < keylen; i++)
temp[i] = input[i];
for(i = 0; i < msglen; i++)
{
quot[i] = temp[0];
if(quot[i] == '0')
for (j = 0; j < keylen; j++)
key[j]='0';
else
{
for (j=0;j<keylen;j++)
key[j]=key1[j];
for (j=keylen-1;j>0;j--)
{
if(temp[j]==key[j])
rem[j-1]='0'; else
rem[j-1]='1';
}
}
rem[keylen-1]=input[i+keylen];
strcpy(temp,rem);
}
strcpy(rem, temp);
printf("\nQuotient is ");
for (i = 0; i < msglen; i++)
printf("%c",quot[i]);
printf("\nRemainder is ");
for(i = 0; i < keylen-1; i++)
printf("%c", rem[i]);
printf("\nFinal data is: ");
for(i = 0; i < msglen; i++)
printf("%c", input[i]);
for (i = 0; i < keylen-1; i++)
printf("%c",rem[i]);
}
```

- **OUTPUT:**

```
Enter Frame size: 4

Enter Frame:0 1 1 1 1

Enter Generator size:
Enter Generator:111

Sender Side:
Frame: 0111
Generator :0
Number of 0's to be appended: 0
Message after appending 0's :0111
CRC bits:
Transmitted Frame: 0111
Receiver side :
Received Frame: 0111
Reaminder:
Since Remainder Is 0 Hence Message Transmitted From Sender To Receriver Is Corr
ect
```

- **DISCUSSION:**

A CRC-enabled device calculates a short, fixed-length binary sequence, known as the check value or CRC, for each block of data to be sent or stored and appends it to the data, forming a codeword. When a codeword is received or read, the device either compares its check value with one freshly calculated from the data block, or equivalently, performs a CRC on the whole codeword and compares the resulting check value with an expected residue constant. If the CRC values do not match, then the block contains a data error. The device may take corrective action, such as rereading the block or requesting that it be sent again. Otherwise, the data is assumed to be error-free (though, with some small probability, it may contain undetected errors; this is inherent in the nature of error-checking).

- **FINDING AND LEARNINGS:**

CRCs are specifically designed to protect against common types of errors on communication channels, where they can provide quick and reasonable assurance of the integrity of messages delivered. However, they are not suitable for protecting against intentional alteration of data. CRC is an easily reversible function, which makes it unsuitable for use in digital signatures. Thirdly, CRC is a linear function with a property that

CRC (X xor Y xor Z) = CRC (X) xor CRC(Y) xor CRC(Z)

As a result, even if the CRC is encrypted with a stream cipher that uses XOR as its combining operation (or mode of block cipher which effectively turns it into a stream cipher, such as OFB or CFB), both the message and the associated CRC can be manipulated without knowledge of the encryption key.

Experiment 3

- **OBJECTIVE:** Write a program to implement stop and wait protocol.

- **THEORY:** Stop-and-wait ARQ, also referred to as alternating bit protocol, is a method in telecommunications to send information between two connected devices. It ensures that information is not lost due to dropped packets and that packets are received in the correct order. It is the simplest automatic repeat-request (ARQ) mechanism. A stop-and-wait ARQ sender sends one frame at a time; it is a special case of the general sliding window protocol with transmit and receive window sizes equal to one and greater than one respectively. After sending each frame, the sender doesn't send any further frames until it receives an acknowledgement (ACK) signal. After receiving a valid frame, the receiver sends an ACK. If the ACK does not reach the sender before a certain time, known as the timeout, the sender sends the same frame again. The timeout countdown is reset after each frame transmission. The above behaviour is a basic example of Stop-and-Wait. However, real-life implementations vary to address certain issues of design.

- **ALGORITHM:**
  Simple Stop and Wait
  Sender:
  Rule 1) Send one data packet at a time.
  Rule 2) Send next packet only after receiving acknowledgement for previous.
  Receiver:
  Rule 1) Send acknowledgement after receiving and consuming of data packet.
  Rule 2) After consuming packet acknowledgement need to be sent (Flow Control)

  There are 3 problems to this:-
  1. Lost Data
  2. Lost Acknowledgement
  3. Delayed acknowledgement

  Stop and Wait ARQ (Automatic Repeat Request)
  Above 3 problems are resolved by Stop and Wait ARQ (Automatic Repeat Request) that does both error control and flow control.

  Working of Stop and Wait ARQ:-
  1. Sender A sends a data frame or packet with sequence number 0.
  2. Receiver B, after receiving data frame, sends and acknowledgement with sequence number 1 (sequence number of next expected data frame or packet)

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

- **CODE:**

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
```

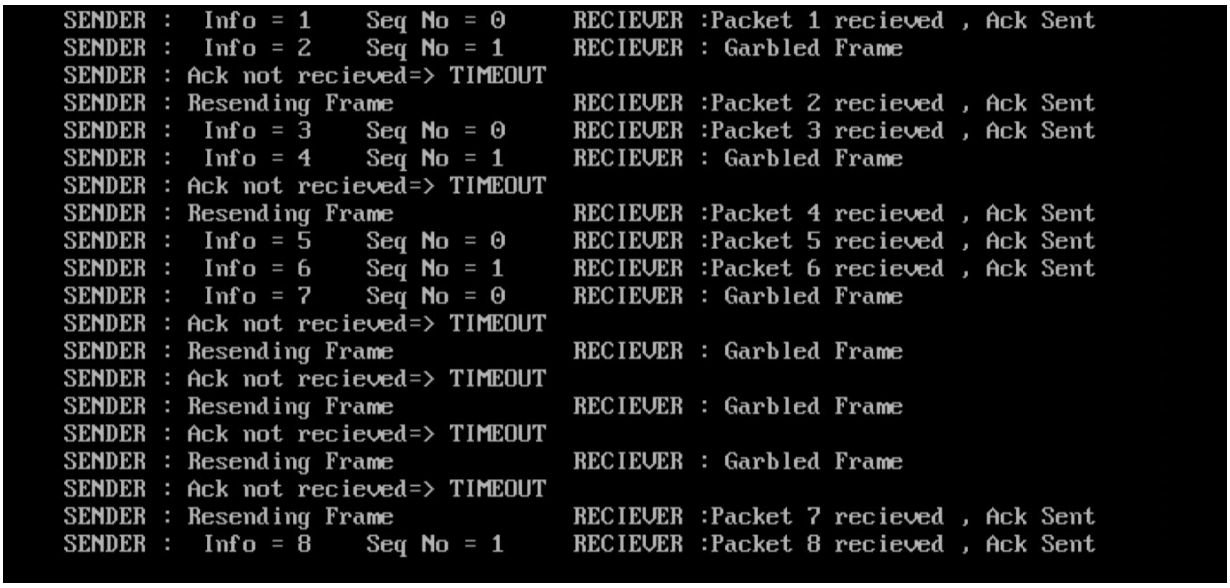
```
#include <time.h>
#define TIMEOUT 5
#define MAX_SEQ 1
#define TOT_PACKETS 8
#define inc(k) if(k<MAX_SEQ) k++; else k=0;
typedef struct
{
int data;
}packet;
typedef struct
{
int kind;
int seq;
int ack;
packet info;
int err;
}frame;
frame DATA;
typedef enum{frame_arrival,err,timeout,no_event} event_type;
void from_network_layer(packet *);
void to_network_layer(packet *);
void to_physical_layer(frame *);
void from_physical_layer(frame *);
void wait_for_event_sender(event_type *);
void wait_for_event_reciever(event_type *);
void reciever();
void sender();
int i=1; //Data to be sent by sender
char turn; //r , s
int DISCONNECT=0;
int main()
{
srand(time(0));
while(!DISCONNECT)
{
sender();
sleep(4);
reciever();
}
}
void sender()
{
static int frame_to_send=0;
static frame s;
packet buffer;
```

```
event_type event;
static int flag=0;
if(flag==0)
{
from_network_layer(&buffer);
s.info = buffer;
s.seq = frame_to_send;
printf("SENDER : Info = %d Seq No = %d ",s.info.data,s.seq);
turn = 'r';
to_physical_layer(&s);
flag = 1;
}
wait_for_event_sender(&event);
if(turn=='s')
{
if(event==frame_arrival)
{
from_network_layer(&buffer);
inc(frame_to_send);
s.info = buffer;
s.seq = frame_to_send;
printf("SENDER : Info = %d Seq No = %d ",s.info.data,s.seq);
turn = 'r';
to_physical_layer(&s);
}
if(event==timeout)
{
printf("SENDER : Resending Frame ");
turn = 'r';
to_physical_layer(&s);
}}
}
void reciever()
{
static int frame_expected=0;
frame r,s;
event_type event;
wait_for_event_reciever(&event);
if(turn=='r')
{
if(event==frame_arrival)
{
from_physical_layer(&r);
if(r.seq==frame_expected)
{
```

```
to_network_layer(&r.info);
inc(frame_expected);
}
else
printf("RECIEVER : Acknowledgement Resent\n");
turn = 's';
to_physical_layer(&s);
}
if(event==err)
{
printf("RECIEVER : Garbled Frame\n");
turn = 's'; //if frame not recieved
} //sender shold send it again
}
}
void from_network_layer(packet *buffer)
{
(*buffer).data = i;
i++;
}
void to_physical_layer(frame *s)
{ // 0 means error
s->err = rand()%4; //non zero means no error
DATA = *s; //probability of error = 1/4
}
void to_network_layer(packet *buffer)
{
printf("RECIEVER :Packet %d recieved , Ack Sent\n",(*buffer).data);
if(i>TOT_PACKETS) //if all packets recieved then disconnect
{
DISCONNECT = 1;
printf("\nDISCONNECTED");
}
}
void from_physical_layer(frame *buffer)
{
*buffer = DATA;
}
void wait_for_event_sender(event_type * e)
{
static int timer=0;
if(turn=='s')
{
timer++;
if(timer==TIMEOUT)
```

```
{
*e = timeout;
printf("SENDER : Ack not recieved=> TIMEOUT\n");
timer = 0;
return;
}
if(DATA.err==0)
*e = err;
else
{
timer = 0;
*e = frame_arrival;
}
}
}
void wait_for_event_reciever(event_type * e)
{
if(turn=='r')
{
if(DATA.err==0)
*e = err;
else
*e = frame_arrival;
}
}
```

- **OUTPUT:**



- **DISSCUSION:**

Brief Description:

• Used in Connection-oriented communication.

• It offers error and flow control

• It is used in Data Link and Transport Layers

• Stop and Wait ARQ mainly implements Sliding Window Protocol concept with Window Size 1

Characteristics of Stop and Wait ARQ:

• It uses link between sender and receiver as half duplex link

• Throughput = 1 Data packet/frame per RTT

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

• It is an example for “Closed Loop OR connection oriented “ protocols

• It is an special category of SWP where its window size is 1

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

- **FINDING AND LEARNINGS:**

The Stop and Wait ARQ solves main three problems, but may cause big performance issues as sender always waits for acknowledgement even if it has next packet ready to send. Consider a situation where you have a high bandwidth connection and propagation delay is also high (you are connected to some server in some other country though a high speed connection). To solve this problem, we can send more than one packet at a time with a larger sequence numbers. So Stop and Wait ARQ may work fine where propagation delay is very less for example LAN connections, but performs badly for distant connections like satellite connection.

Experiment 4

- **OBJECTIVE:** Write a program to implement Sliding Window Protocol

- **THEORY:** Sliding window protocols are data link layer protocols for reliable and sequential delivery of data frames. In this protocol, multiple frames can be sent by a sender at a time before receiving an acknowledgment from the receiver. In the sliding window technique, each data packet (for most data link layers) and byte (in TCP) includes a unique consecutive sequence number, which is used by the receiving computer to place data in the correct order. The objective of the sliding window technique is to use the sequence numbers to avoid duplicate data and to request missing data.

- **CODE:**

```
#include<stdio.h>
int main()
{
int w,i,f,frames[50];
printf("Enter window size: ");
scanf("%d",&w);
printf("\nEnter number of frames to transmit: ");
scanf("%d",&f);
printf("\nEnter %d frames: ",f);
for(i=1;i<=f;i++)
scanf("%d",&frames[i]);
printf("\nWith sliding window protocol the frames will be sent in the following manner
(assuming no corruption of frames)\n\n");
printf("After sending %d frames at each stage sender waits for acknowledgement sent by the
receiver\n\n",w);
for(i=1;i<=f;i++)
{
if(i%w==0)
{
printf("%d\n",frames[i]);
printf("Acknowledgement of above frames sent is received by sender\n\n");
}
else
printf("%d ",frames[i]);
}
if(f%w!=0)
printf("\nAcknowledgement of above frames sent is received by sender\n");
return 0;
}
```

- **OUTPUT**

```
ENTER THE WINDOWS SIZE : 4

SENDER WINDOW IS EXPANDED TO STORE MESSAGE OR WINDOW

ENTER THE DATA TO BE SENT: 1

MESSAGE SEND BY THE SENDER:
1

WINDOW SIZE OF RECEIVER IS EXPANDED

ACKNOWLEDGEMENT FROM RECEIVER

ACK:4
MESSAGE RECEIVED BY RECEIVER IS : 1

WINDOW SIZE OF RECEIVER IS SHRINKED
```

- **DISCUSSION:**

The essence of all sliding window protocols is that at any instant of time, the sender maintains a set of sequence numbers corresponding to frames it is permit-ted to send. These frames are said to fall within the sending window. Similarly, the receiver also maintains a receiving window corresponding to the set of frames it is permitted to accept. The sender's window and the receiver's window need not have the same lower and upper limits or even have the same size. In some protocols they are fixed in size, but in others they can grow or shrink over the course of time as frames are sent and received.

Although these protocols give the data link layer more freedom about the order in which it may send and receive frames, we have definitely not dropped the requirement that the protocol must deliver packets to the destination network layer in the same order they were passed to the data link layer on the sending machine.

- **FINDING AND LEARNINGS:**

Sliding window is a theoretical concept and there are several different implementations of this concept based on sender and receiver's window sizes.

Two popular protocols using sliding window concept are:

1. Go Back N (GBN)
2. Selective Repeat (SR)

Go-Back N

Sender Window Size (WS)

It is N itself. If we say the protocol is GB10, then Ws = 10. N should be always greater than 1 in order to implement pipelining. For N = 1, it reduces to Stop and Wait protocol.
Efficiency Of GBN = N/(1+2a)
Where a = Tp/Tt
If B is the bandwidth of the channel, then
Effective Bandwidth or Throughput = Efficiency * Bandwidth = (N/(1+2a)) * B.
Receiver Window Size (WR)
WR is always 1 in GBN.
Acknowledgements
There are 2 kinds of acknowledgements namely :
• Cumulative Ack.
• Independent Ack

Selective Repeat
This protocol(SRP) is mostly identical to GBN protocol, except that buffers are used and the receiver, and the sender, each maintain a window of size. SRP works better when the link is very unreliable. Because in this case, retransmission tends to happen more frequently, selectively retransmitting frames is more efficient than retransmitting all of them. SRP also requires full duplex link.
Sender's Windows (Ws) = Receiver's Windows (Wr).
Efficiency of Selective Repeat Protocol (SRP) is same as GO-Back-N's efficiency :
Efficiency = N/(1+2a)
Where a = Propagation delay / Transmission delay
Buffers = N + N
Sequence number = N(sender side) + N (Receiver Side)

Experiment 5

- **OBJECTIVE:** Write a program to display the Network Class, Network ID and Host ID of a given IP Address

- **THEORY:** Given a valid IPv4 address in the form of string and it follows Class Full addressing. The 32 bit IP address is divided into five subclasses. These are:
  1. Class A
  2. Class B
  3. Class C
  4. Class D
  5. Class E

  Each of these classes has a valid range of IP addresses. Classes D and E are reserved for multicast and experimental purposes respectively.
  1. For determining the Class: The idea is to check first octet of IP address. As we know, for class At first octet will range from 1 – 126, for class B first octet will range from 128 – 191, for class C first octet will range from 192- 223, for class D first octet will range from 224 – 239, for class E first octet will range from 240 – 255.
  2. For determining the Network and Host ID: We know that Subnet Mask for Class A is 8, for Class Bis 16 and for Class C is 24 whereas Class D and E is not divided into Network and Host ID.

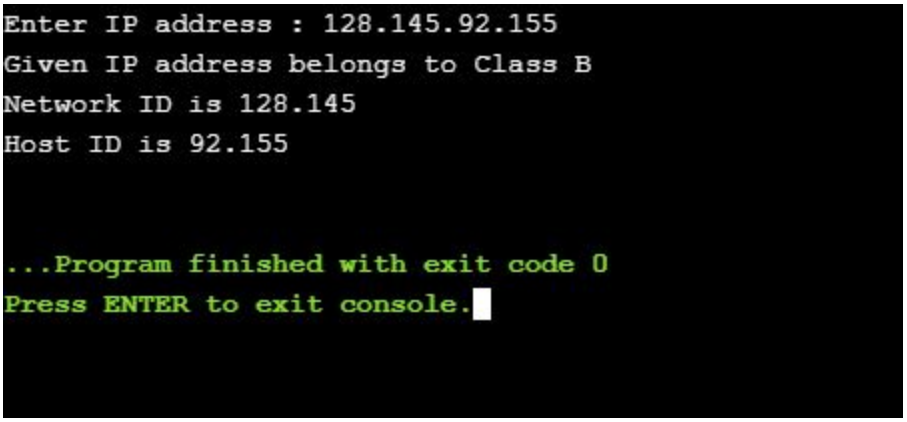
- **CODE**

```
#include<stdio.h>
#include<string.h>
char findClass(char str[])
{
char arr[4];
int i = 0;
while (str[i] != '.')
{
arr[i] = str[i];
i++;
}
i--;
int ip = 0, j = 1;
while (i >= 0)
{
ip = ip + (str[i] - '0') * j;
j = j * 10;
i--;
}
if (ip >=0 && ip <= 127)
return 'A';
else if (ip >= 128 && ip <= 191)
return 'B';
```

```
else if (ip >= 192 && ip <= 223)
return 'C';
else if (ip >= 224 && ip <= 239)
return 'D';
else
return 'E';
}
void separate(char str[], char ipClass)
{
char network[12], host[12];
for (int k = 0; k < 12; k++)
network[k] = host[k] = '\0';
if (ipClass == 'A')
{
int i = 0, j = 0;
while (str[j] != '.')
network[i++] = str[j++];
i = 0;
j++;
while (str[j] != '\0')
host[i++] = str[j++];
printf("Network ID is %s\n", network);
printf("Host ID is %s\n", host);
}
else if (ipClass == 'B')
{
int i = 0, j = 0, dotCount = 0;
while (dotCount < 2)
{
network[i++] = str[j++];
if (str[j] == '.')
dotCount++;
}
i = 0;
j++;
while (str[j] != '\0')
host[i++] = str[j++];
printf("Network ID is %s\n", network);
printf("Host ID is %s\n", host);
}
else if (ipClass == 'C')
{
int i = 0, j = 0, dotCount = 0;
while (dotCount < 3)
{
```

```
network[i++] = str[j++];
if (str[j] == '.')
dotCount++;
}
i = 0;
j++;
while (str[j] != '\0')
host[i++] = str[j++];
printf("Network ID is %s\n", network);
printf("Host ID is %s\n", host);
}
else
printf("In this Class, IP address is not"
" divided into Network and Host ID\n");
}
int main()
{
char str[50];
printf("Enter IP address : ");
scanf("%s", str);
char ipClass = findClass(str);
printf("Given IP address belongs to Class %c\n",
ipClass);
separate(str, ipClass);
return 0;
}
```

- **OUTPUT:**



- **DISCUSSION:**

  IPv4 address is divided into two parts:
  • Network ID
  • Host ID
  The class of IP address is used to determine the bits used for network ID and host ID and the number of total networks and hosts possible in that particular class. Each ISP or network administrator assigns IP address to each device that is connected to its network.

- **FINDING AND LEARNINGS**

  1. Class A
  The network ID is 8 bits long. The host ID is 24 bits long. The higher order bit of the first octet in class A is always set to 0. The remaining 7 bits in first octet are used to determine network ID. The 24 bits of host ID are used to determine the host in any network. The default sub-net mask for class A is 255.x.x.x. Therefore, class A has a total of: 2^7= 128 network ID and 2^24 – 2 = 16,777,214 host ID
  IP addresses belonging to class A ranges from 1.x.x.x – 126.x.x.x

  2. Class B
  The network ID is 16 bits long. • The host ID is 16 bits long. The higher order bits of the first octet of IP addresses of class B are always set to 10. The remaining 14 bits are used to determine network ID. The 16 bits of host ID is used to determine the host in any network. The default sub-net mask for class B is 255.255.x.x. Class B has a total of: • 2^14 = 16384 network address • 2^16 – 2 = 65534 host address
  IP addresses belonging to class B ranges from 128.0.x.x – 191.255.x.x.

  3. Class C
  The network ID is 24 bits long.
  • The host ID is 8 bits long. The higher order bits of the first octet of IP addresses of class C are always set to 110. The remaining 21 bits are used to determine network ID. The 8 bits of host ID is used to determine the host in any network. The default sub-net mask for class C is 255.255.255.x. Class C has a total of:
  • 2^21 = 2097152 network address • 2^8 – 2 = 254 host address
  IP addresses belonging to class C ranges from 192.0.0.x – 223.255.255.x.

  4. Class D
  IP address belonging to class D are reserved for multi-casting. The higher order bits of the first octet of IP addresses belonging to class D are always set to 1110. The remaining bits are for the address that interested hosts recognize. There are no host and network ID as present with Class A, B and C. Class D does not possess any sub-net mask. IP addresses belonging to class D ranges from 224.0.0.0 – 239.255.255.255.

  5. Class E
  IP addresses belonging to class E are reserved for experimental and research purposes. IP addresses of class E ranges from 240.0.0.0 – 255.255.255.255. This class doesn't have any subnet mask. The higher order bits of first octet of class E are always set to 1111.

Experiment- 6

- **OBJECTIVE** :Write a program to implement Distance Vector Routing Algorithm

- **THEORY**: A distance-vector routing algorithm also called the Bellman-Ford algorithm is a routing algorithm in which every router maintains a database with one entry for each possible destination on the network. It requires that a router inform its neighbours of topology changes periodically. Routers select the route with the lowest cost to each possible destination and add this to their own routing tables. These neighbors propagate the information to their neighbors hop by hop until information from all routers has spread throughout the entire internetwork.

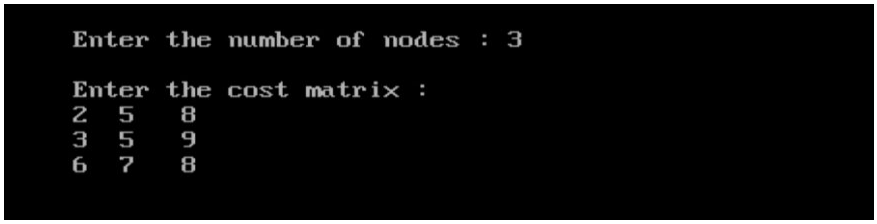
- **ALGORITHM**:
  1. A router transmits its distance vector to each of its neighbours in a routing packet.
  2. Each router receives and saves the most recently received distance vector from each of its neighbours.
  3. A router recalculates its distance vector when:
  - It receives a distance vector from a neighbour containing different information than before.
  - It discovers that a link to a neighbour has gone down.
  Example – In this network we have 5 routers A, B, C, D and E:
  Router A will share it routing table to neighbours and neighbours will share it routing table to it to A and distance from node A to destination will be calculated using Bellmen-Ford equation:
  $D(A,B) = \min \{ C(A,B) + Dv(B)\}$ for each node $B \in N$
  After n-1 iterations, the routing table converges and all the nodes have shortest route to every other node.

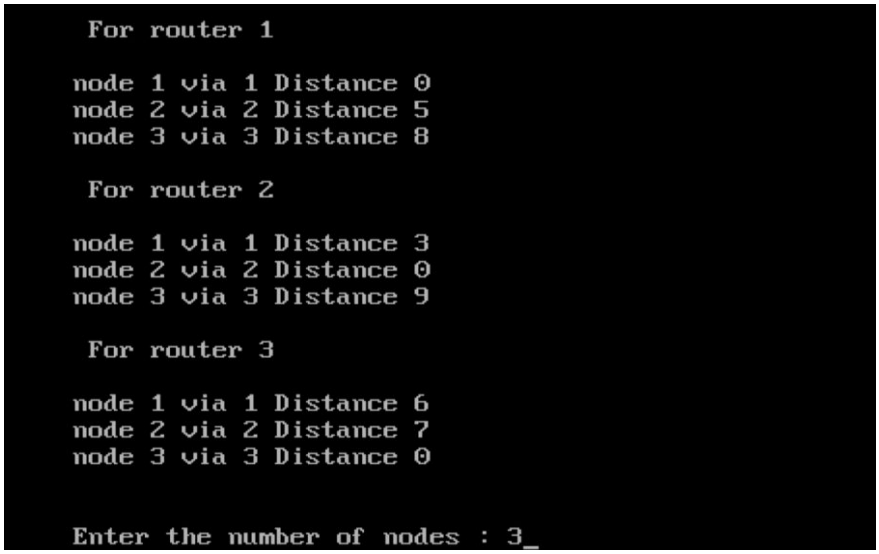
- **CODE:**

```
#include<stdio.h>
struct node
{
unsigned dist[20];
unsigned parent[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);
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];
```

```
rt[i].parent[j]=j; }
do
{
count=0;
for(i=0;i<nodes;i++)
for(j=0;j<nodes;j++)
for(k=0;k<nodes;k++)
if(rt[i].dist[j]>costmat[i][k]+rt[k].dist[j])
{
rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];
rt[i].parent[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].parent[j]+1,rt[i].dist[j]);
}
printf("\n\n");
}
```

- **OUTPUT:**





- **DISCUSSION:**

  Distance vector algorithm is a dynamic algorithm. Each of the routers present in the network maintains a distance vector to store information regarding the shortest route to any given node from that node. The Distance vector algorithm is iterative, asynchronous and distributed.

  1. Distributed: It is distributed in that each node receives information from one or more of its directly attached neighbours, performs calculation and then distributes the result back to its neighbours.
  2. Iterative: It is iterative in that its process continues until no more information is available to be exchanged between neighbours.
  3. Asynchronous: It does not require that all of its nodes operate in the lock step with each other.

- **FINDING AND LEARNINGS**

  Advantages of Distance Vector routing :-

  • It is easy to administer and implement.
  • It is simpler to configure and maintain than link state routing.
  • Requires less hardware and processing power than other routing methods.

  Disadvantages of Distance Vector routing :-

  • The problem with distance vector routing is its slowness in converging to the correct answer. This is due to a problem called count to infinity problem.
  • Another problem is that this algorithm does not take the line bandwidth into consideration when choosing root. For larger networks, distance vector routing results in larger routing tables than link state since each router must know about all other routers. This can also lead to congestion on WAN links.
  • It creates more traffic than link state since a hop count change must be propagated to all routers and processed on each router. Hop count updates take place on a periodic basis, even if there are no changes in the network topology, so bandwidth-wasting broadcasts still occur.

Experiment 7

- **OBJECTIVE:** Write a program to implement Link State Routing Algorithm

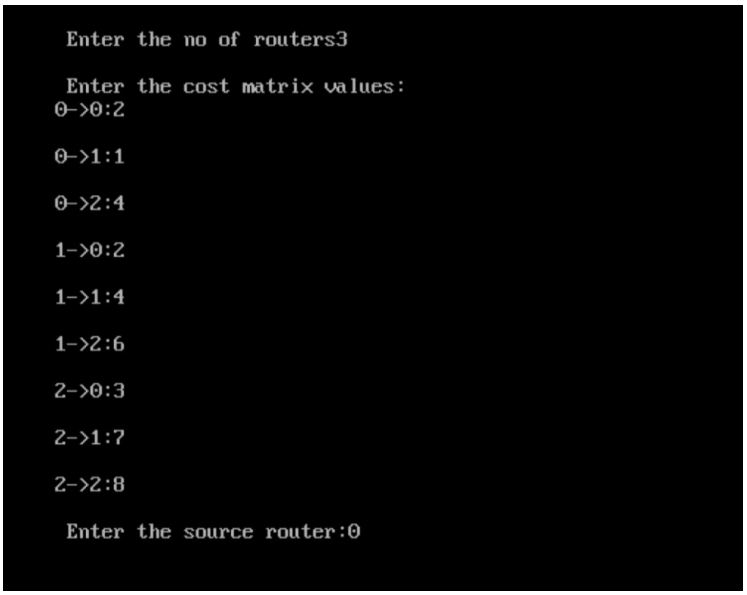
- **THEORY:** Link state routing is the second family of routing protocols. While distance vector routers use a distributed algorithm to compute their routing tables, link-state routing uses link-state routers to exchange messages that allow each router to learn the entire network topology. Based on this learned topology, each router is then able to compute its routing table by using a shortest path computation.
  Features of link state routing protocols –
  • Link state packet – A small packet that contains routing information.
  • Link state database – A collection information gathered from link state packet.
  • Shortest path first algorithm (Dijkstra algorithm) – A calculation performed on the database results into shortest path. Routing table – A list of known paths and interfaces.

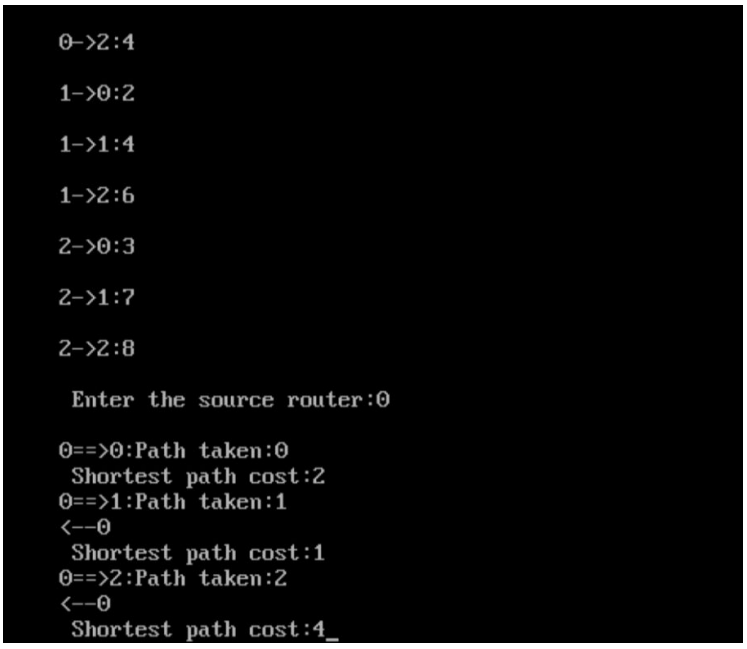
- **CODE**

```
#include <stdio.h>
#include <string.h>
int main()
{
int count,src_router,i,j,k,w,v,min;
int cost_matrix[100][100],dist[100],last[100];
int flag[100];
printf("\n Enter the no of routers : ");
scanf("%d",&count);
printf("\n Enter the cost matrix values : \n");
for(i=0;i<count;i++)
{
for(j=0;j<count;j++)
{
printf("\n %d to %d: ",i,j);
scanf("%d",&cost_matrix[i][j]);
if(cost_matrix[i][j]<0)cost_matrix[i][j]=1000;
}
}
printf("\n Enter the source router:");
scanf("%d",&src_router);
for(v=0;v<count;v++)
{
flag[v]=0;
last[v]=src_router;
dist[v]=cost_matrix[src_router][v];
}
flag[src_router]=1;
for(i=0;i<count;i++)
{
```

```
min=1000;
for(w=0;w<count;w++)
{
if(!flag[w])
if(dist[w]<min)
{
v=w;
min=dist[w];
}
}
flag[v]=1;
for(w=0;w<count;w++)
{
if(!flag[w])
if(min+cost_matrix[v][w]<dist[w])
{
dist[w]=min+cost_matrix[v][w];
last[w]=v;
}
}
}
for(i=0;i<count;i++)
{
printf("\n %d to %d:Path taken: %d",src_router,i,i);
w=i;
while(w!=src_router)
{
printf(" <-- %d",last[w]);w=last[w];
}
printf("\n Shortest path cost:%d \n",dist[i]);
}
}
```

- **OUTPUT:**





- **DISCUSSION:**

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

• Step-1: The node is taken and chosen as a root node of the tree, this creates the tree with a single node, and now set the total cost of each node to some value based on the information in Link State Database

• Step-2: Now the node selects one node, among all the nodes not in the tree like structure, which is nearest to the root, and adds this to the tree.The shape of the tree gets changed .

• Step-3: After this node is added to the tree, the cost of all the nodes not in the tree needs to be updated because the paths may have been changed.

• Step-4: The node repeats the Step 2. and Step 3. until all the nodes are added in the tree.

- **FINDING AND LEARNINGS**

Open shortest path first (OSPF) routing protocol –

• Open Shortest Path First (OSPF) is a unicast routing protocol developed by working group of the Internet Engineering Task Force (IETF).

• It is a intradomain routing protocol.

• It is an open source protocol.

• It is similar to Routing Information Protocol (RIP)

•OSPF is a classless routing protocol,.This provides network administrators with extra network-configuration flexibility.These updates are multicasts at specific addresses (224.0.0.5 and 224.0.0.6).

• OSPF is implemented as a program in the network layer using the services provided by the Internet Protocol

• IP datagram that carries the messages from OSPF sets the value of protocol field to 89

• OSPF is based on the SPF algorithm, which sometimes is referred to as the Dijkstra algorithm

Link State protocols in comparison to Distance Vector protocols have:

1. It requires large amount of memory.
2. Shortest path computations require many CPU circles.
3. If network use the little bandwidth ; it quickly reacts to topology changes
4. All items in the database must be sent to neighbors to form link state packets.
5. All neighbors must be trusted in the topology.