Computer Networks Lab

(B20CS3109)

Lab Manual

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SRKR ENGINEERING COLLEGE (A)

Chinna Amiram, Bhimavaram, West Godavari Dist., A.P.

COURSE OUTCOMES

CO1-Implement datalink layer framing methods like error control and flow control.
CO2-Examines and implement the various Routing algorithms.
CO3-Develop client-server applications using sockets.

INDEX

Data Link Layer

Data Link Layer is the second layer in the OSI reference model. It represents creating a shared transmission medium and frequent transmission of the data frame in a computer transmission setting.

It can access a natural flow of elements for the physical layer at the sender device. The basic flow of data is generated using multiple technologies such as cable, DSL, wireless, optical fibre, etc.

Services Provided to Data Link Layer

The primary service of the data link layer is to support error-free transmission. The physical layer sends the data from the sender's hub to the receiver's hub as raw bits. The data link layer should recognize and correct some errors in the communicated data.

The data link layer provides a distinct connection to the network layer. It is used to handle communication bugs, control the data stream, and manage sender and receiver inconsistency by maintaining the multiple services. It can work these actions in the following method −

Unacknowledged connectionless service − This contains separate frames from the source host to the destination host without some acknowledgment structure. It does not have any link established or launched. It does not manage with frame recovery due to channel noise.

Acknowledged connectionless service − The transmission medium is more error-prone. This requires acceptance service for each frame shared between two hosts to provide that the frame has occurred correctly.

Acknowledged connection-oriented service − This layer supports this service to the network layer by settling a link between the source and destination hosts before any information removal occurs.

Framing − In this layer, it receives a raw bitstream from the physical layer that cannot be bug-free. The data link layer divides the bitstreams into frames to provide a frequent change of bitstreams to the network layer.

Error Control − It includes sequencing frames and sending control frames for acceptance. A noisy channel can avoid scanning of bits, falling bits from a frame, introducing specific bits in the frame, frames final sinking, etc.

Flow Control − There is another fundamental problem in the data link design to regulate the cost of data communication between two source and destination hosts. If the conflict among the source and destination hosts data sending and receiving speed, it will create packets to drop at the receiver end.

Sequence Integrity − The data link layer supports the data bits sequence and sends them to the physical layer in the similar sequence as received from the network layer. It supports a reliable share of data link service data unit (DLSDU) over the data link connections.

Bit Stuffing & Character Stuffing

Bit stuffing is the mechanism of inserting one or more non-information bits into a message to be transmitted, to break up the message sequence, for synchronization purpose.

Byte stuffing is a mechanism to convert a message formed of a sequence of bytes that may contain reserved values such as frame delimiter, into another byte sequence that does not contain the reserved values.

Purposes of byte stuffing and bit stuffing

In Data Link layer, the stream of bits from physical layer are divided into data frames. The data frames can be of fixed length or variable length. In variable - length framing, the size of each frame to be transmitted may be different. So, a pattern of bits is used as a delimiter to mark the end of one frame and the beginning of the next frame. However, if the pattern occurs in the message, then mechanisms needs to be incorporated so that this situation is avoided.

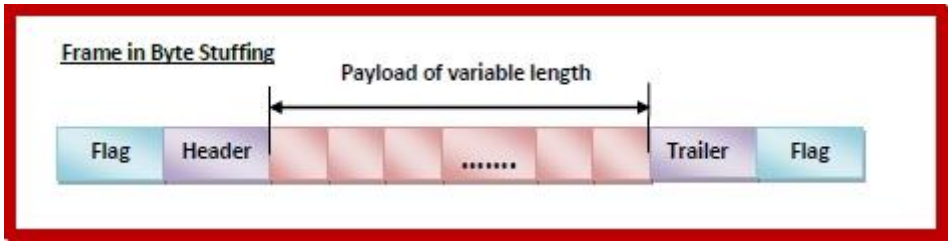
The two common approaches are −

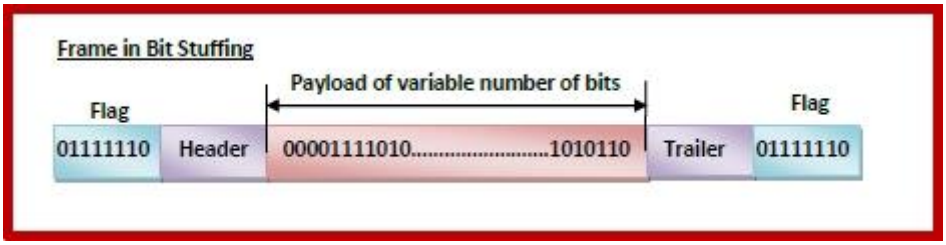
- Byte - Stuffing − A byte is stuffed in the message to differentiate from the delimiter. This is also called character-oriented framing.
- Bit - Stuffing − A pattern of bits of arbitrary length is stuffed in the message to differentiate from the delimiter. This is also called bit - oriented framing.

Data link layer frames in byte stuffing and bit stuffing

A data link frame has the following parts −

- Frame Header − It contains the source and the destination addresses of the frame.
- Payload field − It contains the message to be delivered. In bit stuffing it is a variable sequence of bits, while in byte stuffing it is a variable sequence of data bytes.
- Trailer − It contains the error detection and error correction bits.
- Flags − Flags are the frame delimiters signalling the start and end of the frame. In bit stuffing, flag comprises of a bit pattern that defines the beginning and end bits. It is generally of 8-bits and comprises of six or more consecutive 1s. In byte stuffing, flag is of 1- byte denoting a protocol - dependent special character.





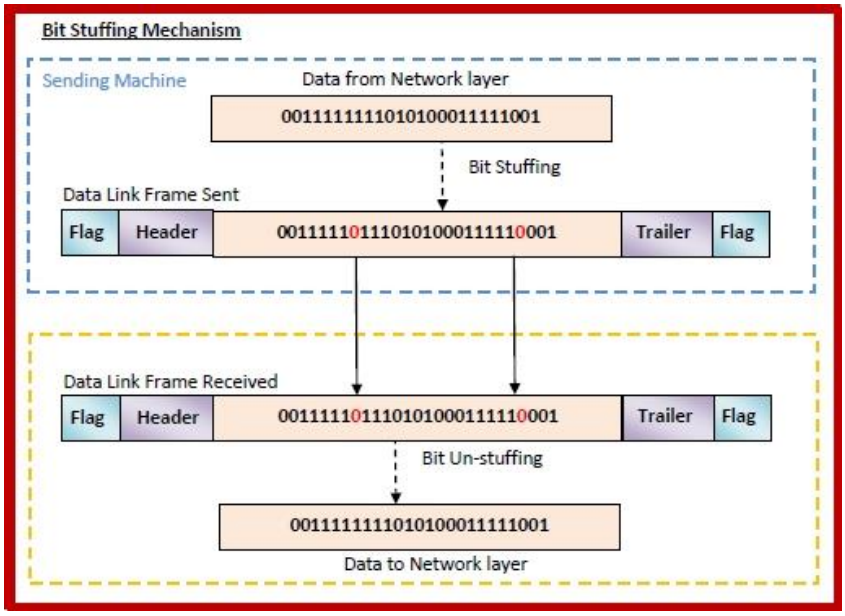
Mechanisms of bit stuffing versus byte stuffing

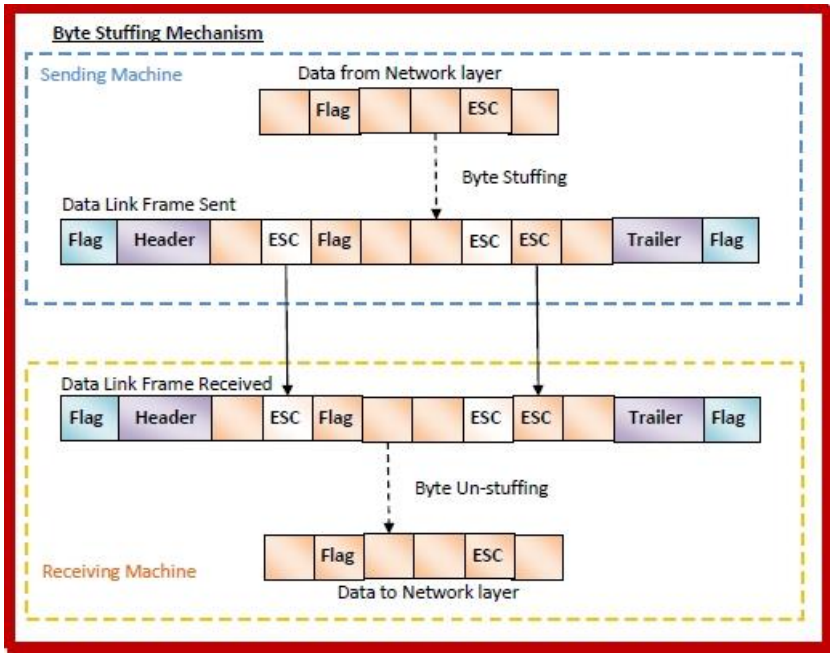
Bit Stuffing Mechanism

Here, the delimiting flag sequence generally contains six or more consecutive 1s. Most protocols use the 8-bit pattern 01111110 as flag. In order to differentiate the message from the flag in case of same sequence, a single bit is stuffed in the message. Whenever a 0 bit is followed by five consecutive 1bits in the message, an extra 0 bit is stuffed at the end of the five 1s. When the receiver receives the message, it removes the stuffed 0s after each sequence of five 1s. The un-stuffed message is then sent to the upper layers.

Byte Stuffing Mechanism

If the pattern of the flag byte is present in the message byte sequence, there should be a strategy so that the receiver does not consider the pattern as the end of the frame. Here, a special byte called the escape character (ESC) is stuffed before every byte in the message with the same pattern as the flag byte. If the ESC sequence is found in the message byte, then another ESC byte is stuffed before it.





Implementation of bit stuffing and character stuffing

Description:

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.).

Bit sequence: 11010111110101111101011111110 (without bit stuffing)

Bit sequence: 11010111110010111110101011111010 (with bit stuffing)

After 5 consecutive 1-bits, a 0-bit is stuffed.

Program:

```
#include<stdio.h>
int main()
{
    int a[30],b[30],i,c=0,k=0,j,r=0;
    long int w;
    printf("Enter input : ");
    scanf("%ld",&w);
    for (i=0;i<20;i++)
    {    if(w!=0)
        {
            b[i]=w%10;
            w=w/10;
            r++;
        }    }
    for (i=r-1;i>=0;i--)
    {
```

```
        a[k]=b[i];
        k++;     }
    k=0;
     for (i=0;i<20;i++)
    {
        if(a[i]==1)
        {
            c++;
        }
        if(a[i]==0)
        {
            c=0;
        }
        if(c==6)
        {
            for(j=29;j>=i;j--)
            {
                a[j+1]=a[j];
            }
            a[i]=0;
            k++;
            c=0;
        }
    }
    printf("After Bit stuffing : ");
    for (i=0;i<r+k;i++)
    {
        printf("%d",a[i]);
    }
  printf("\n");
}
```

Output:

```
lab5@lab5-OptiPlex-3020:~/Desktop$ gcc bitstuf.c
lab5@lab5-OptiPlex-3020:~/Desktop$ ./a.out
Enter input : 1010111111
After Bit stuffing : 10101111101
lab5@lab5-OptiPlex-3020:~/Desktop$
```

Bit Destuffing:

Description:
It is the complete opposite of Bit stuffing you are given an array of 0's and 1's and if there are five consecutive 1's followed by a 0, you have to remove o from the array.
Bit sequence: 10101111101 (Before Bit Destuffing)
Bit sequence: 1010111111 (After Bit Destuffing)
Program:

```
#include <stdio.h>
#include <string.h>
void bitDestuffing(int N, int arr[])
{
    int brr[30];
    int i, j, k;
    i = 0;
    j = 0;
    int count = 1;
    while (i < N) {
        if (arr[i] == 1) {
            brr[j] = arr[i];
            for (k = i + 1;
                arr[k] == 1
                && k < N
                && count < 5;
                k++) {
                j++;
                brr[j] = arr[k];
                count++;
                if (count == 5) {
                    k++;                }
                i = k;              }         }
        else {
            brr[j] = arr[i];
        }
        i++;
        j++;
    }

        printf("After Bit De-Stuffing : ");
        for (i = 0; i < j; i++)
```

```
        printf("%d", brr[i]);
    printf("\n");       }
int main()
{
    int N,i,arr[30];
    printf("Enter size : ");
    scanf("%d",&N);
    printf("Enter input : ");
    for(i=0;i<N;i++)
    {
        scanf("%d",&arr[i]);
    }
    bitDestuffing(N, arr);
    return 0;    }
```

Output:

```
lab1@lab1-OptiPlex-5060:~$ gcc destuf.c
lab1@lab1-OptiPlex-5060:~$ ./a.out
Enter size : 10
Enter input : 1 0 1 1 1 1 1 0 1 0
After Bit De-Stuffing : 101111110
lab1@lab1-OptiPlex-5060:~$
```

Character Stuffing:

Description:

Character stuffing is also known as byte stuffing or character-oriented framing and is same as that of bit stuffing but byte stuffing actually operates on bytes whereas bit stuffing operates on bits. In byte stuffing, special byte that is basically known as ESC (Escape Character) that has predefined pattern is generally added to data section of the data stream or frame when there is message or character that has same pattern as that of flag byte.

But receiver removes this ESC and keeps data part that causes some problems or issues. In simple words, we can say that character stuffing is addition of 1 additional byte if there is presence of ESC or flag in text.

Program:

```
#include<stdio.h>
#include<string.h>
int main()
{    char a[30],b[50]="",t[3],start,end,x[3],s[3],d[3],y[3];
    int i,j;
```

```
    printf("Enter characters to be stuffed : ");
    scanf("%s",a);
    printf("\nEnter starting delimiter : ");
    scanf(" %c",&start);
    printf("\nEnter  ending delimiter   : ");
    scanf(" %c",&end);
    x[0]=s[0]=s[1]=start;
    x[1]=s[2]='\0';
    y[0]=d[0]=d[1]=end;
    d[2]=y[1]='\0';
    strcat(b,x);
    for(i=0;i<strlen(a);i++)
    {    t[0]=a[i];
        t[1]='\0';
        if(t[0]==start)
            strcat(b,s);
        else if(t[0]==end)
            strcat(b,d);
        else
            strcat(b,t);     }
   strcat(b,y);
   printf("\nAfter stuffing : %s\n\n",b);
}
```

Output:

```
lab5@lab5-OptiPlex-3020:~/Desktop$ gcc charstuf.c
lab5@lab5-OptiPlex-3020:~/Desktop$ ./a.out
Enter characters to be stuffed : cnlab

Enter starting delimiter : n

Enter  ending delimiter   : a

After stuffing : ncnnlaaba

lab5@lab5-OptiPlex-3020:~/Desktop$
```

Cyclic Redundancy Checks (CRC)

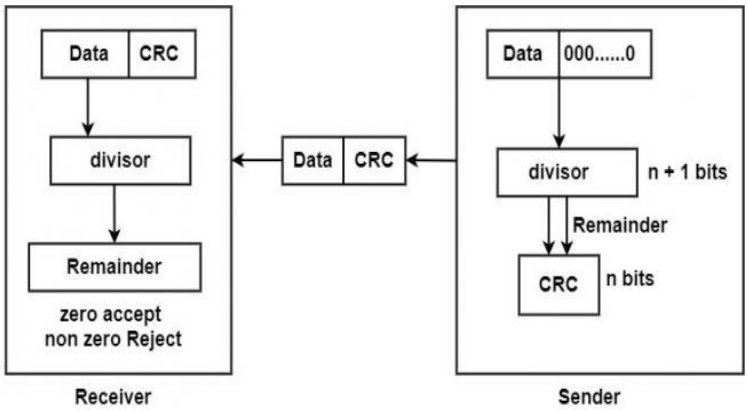
The Cyclic Redundancy Checks (CRC) is the most powerful method for Error-Detection and Correction. It is given as a kbit message and the transmitter creates an (n – k) bit sequence called frame check sequence. The out coming frame, including n bits, is precisely divisible by some fixed number. Modulo 2 Arithmetic is used in this binary addition with no carries, just like the XOR operation.

Redundancy means duplicacy. The redundancy bits used by CRC are changed by splitting the data unit by a fixed divisor. The remainder is CRC.

Qualities of CRC

- It should have accurately one less bit than the divisor.
- Joining it to the end of the data unit should create the resulting bit sequence precisely divisible by the divisor.

CRC generator and checker



Process

- A string of n 0s is added to the data unit. The number n is one smaller than the number of bits in the fixed divisor.
- The new data unit is divided by a divisor utilizing a procedure known as binary division; the remainder appearing from the division is CRC.
- The CRC of n bits interpreted in phase 2 restores the added 0s at the end of the data unit.

Example

Message D = 1010001101 (10 bits)

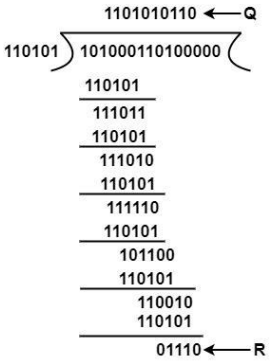
Predetermined P = 110101 (6 bits)

FCS R = to be calculated 5 bits

Hence, n = 15 K = 10 and (n – k) = 5

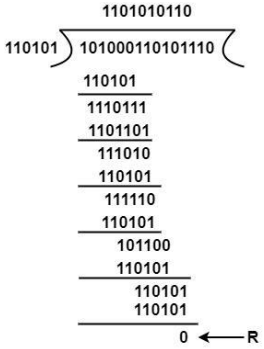
The message is generated through 2^5: accommodating 1010001101000

The product is divided by P.



The remainder is inserted to 2^5D to provide T = 101000110101110 that is sent.

Suppose that there are no errors, and the receiver gets T perfect. The received frame is divided by P.



Because of no remainder, there are no errors.

Implementation of Cyclic Redundancy check (CRC-12, CRC-16, CRC-CCIP).

Description:

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.

Program:

```
#include<stdio.h>
int rem(int,int);
void main()
{    int i,j,k,dl,dil;
    int data[30],div[10],newdata[30],crc[10],datacrc[30],revdata[30],remd[10];
    printf("\n Enter the data length= ");
    scanf("%d",&dl);
    printf("\n Enter the divisor length= ");
    scanf("%d",&dil);
    printf("\n Enter the data : ");
    for(i=0;i<dl;i++)
```

```
        scanf("%d",&data[i]);
    printf("\n Enter the divisor : ");
    for(i=0;i<dil;i++)
        scanf("%d",&div[i]);
    printf("\n The new data is : ");
    for(i=0;i<(dl+dil-1);i++)
    {       if(i<dl)
        newdata[i]=data[i];
        else
        newdata[i]=0;
        printf("%d",newdata[i]);    }
    for(j=0;j<=dl;j++)
    {   for(i=0;i<dil;i++)
        {  crc[i]=newdata[i+j];
            if(crc[0]==1)
            newdata[i+j]=rem(newdata[i+j],div[i]);
            else
            newdata[i+j]=rem(newdata[i+j],0);
        }     printf("\n The Crc is : ");
        for(i=0;i<dil-1;i++)
        printf("%d",crc[i]);     }
    printf("\n The data to be send is : ");
    for(i=0;i<(dl+dil-1);i++)
    {       if(i<dl)
        datacrc[i]=data[i];
        else
        datacrc[i]=crc[i-dl];
        printf("%d",datacrc[i]);    }
//Code for CRC checker:
    printf("\n Enter the receiver side data : ");
    for(i=0;i<(dl+dil-1);i++)
    scanf("%d",&revdata[i]);
    for(j=0;j<=dl;j++)
    {       for(i=0;i<dil;i++)
        {           remd[i]=revdata[i+j];
            if(remd[0]==1)
            revdata[i+j]=rem(revdata[i+j],div[i]);
            else
            revdata[i+j]=rem(revdata[i+j],0);       }
        printf("\n The reminder is : ");
        k=0;
        for(i=0;i<dil-1;i++)
```

```
        {    printf("%d",remd[i]);
             if(remd[i]==0)
             k++;
        }      }
if(k==dil-1)
    printf("\n No error found\n");
    else
    printf("\n There is an error!\n");
}
int rem(int x, int y)
{    if(x==y)
    return 0;
    else
    return 1;    }
```

Output:

CRC-12:

```
lab2@lab2-OptiPlex-3090:~/Desktop$ gcc crc.c
lab2@lab2-OptiPlex-3090:~/Desktop$ ./a.out

 Enter the data length= 12

 Enter the divisor length= 5

 Enter the data : 1 0 1 1 1 0 0 1 1 1 0 1

 Enter the divisor : 1 1 0 0 1

 The new data is : 1011100111010000
 The Crc is : 1011
 The Crc is : 1110
 The Crc is : 0101
 The Crc is : 1010
 The Crc is : 1100
 The Crc is : 0000
 The Crc is : 0001
 The Crc is : 0010
 The Crc is : 0101
 The Crc is : 1010
 The Crc is : 1101
 The Crc is : 0011
 The Crc is : 0110
 The data to be send is : 1011100111010110
 Enter the receiver side data : 1 0 1 1 1 0 0 1 1 1 0 1 0 1 1 0

 The reminder is : 1011
 The reminder is : 1110
 The reminder is : 0101
 The reminder is : 1010
 The reminder is : 1100
 The reminder is : 0000
 The reminder is : 0001
 The reminder is : 0010
 The reminder is : 0101
 The reminder is : 1010
 The reminder is : 1100
 The reminder is : 0000
 The reminder is : 0000
 No error found
lab2@lab2-OptiPlex-3090:~/Desktop$
```

CRC-16:

```
lab2@lab2-OptiPlex-3090:~/Desktop$ gcc crc.c
lab2@lab2-OptiPlex-3090:~/Desktop$ ./a.out

 Enter the data length= 16

 Enter the divisor length= 6

 Enter the data : 1 0 1 1 1 0 1 0 1 1 1 0 0 0 1 0

 Enter the divisor : 1 1 0 0 1 1

 The new data is : 10111010111000100000
 The Crc is : 10111
 The Crc is : 11101
 The Crc is : 01000
 The Crc is : 10000
 The Crc is : 10010
 The Crc is : 10110
 The Crc is : 11110
 The Crc is : 01111
 The Crc is : 11110
 The Crc is : 01111
 The Crc is : 11111
 The Crc is : 01101
 The Crc is : 11010
 The Crc is : 00111
 The Crc is : 01110
 The Crc is : 11100
 The Crc is : 01011
 The data to be send is : 10111010111000100011
 Enter the receiver side data : 1 0 1 1 1 0 1 0 1 1 1 0 0 0 1 0 0 1 0 1 1

 The reminder is : 10111
 The reminder is : 11101
 The reminder is : 01000
 The reminder is : 10000
 The reminder is : 10010
 The reminder is : 10110
 The reminder is : 11110
 The reminder is : 01111
 The reminder is : 11110
 The reminder is : 01111
 The reminder is : 11111
 The reminder is : 01101
 The reminder is : 11010
 The reminder is : 00110
 The reminder is : 01100
 The reminder is : 11001
 The reminder is : 00000
 No error found
lab2@lab2-OptiPlex-3090:~/Desktop$
```

Implementation of sliding window protocol.

Description:

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

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

Programs:

Version 1:

```
#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 frames: ");
    for(i=1;i<=f;i++)
       scanf("%d",&frames[i]);
    printf("After sending %d frames at each stage sender waits for
acknowledgement sent by the receiver:\n",w);
    for(i=1;i<=f;i++)
   {   if(i%w==0)
       {
          printf("%d : ",frames[i]);
          printf("Acknowledgement of above frames sent is received by sender\n");
       }
       else
          printf("%d ",frames[i]);
   }
   if(f%w!=0)
       printf("\nAcknowledgement of above frames sent is received by sender\n");
 }
```

Output:

```
lab2@lab2-OptiPlex-3090:~/Downloads$ gcc slid_ver1.c
lab2@lab2-OptiPlex-3090:~/Downloads$ ./a.out
Enter window size: 4

Enter number of frames to transmit: 20

Enter frames: 1 2 4 5 7 8 9 4 5 6 3 2 5 5 6 4 7 4 5 8
After sending 4 frames at each stage sender waits for acknowledgement sent by the receiver:
1 2 4 5 : Acknowledgement of above frames sent is received by sender
7 8 9 4 : Acknowledgement of above frames sent is received by sender
5 6 3 2 : Acknowledgement of above frames sent is received by sender
5 5 6 4 : Acknowledgement of above frames sent is received by sender
7 4 5 8 : Acknowledgement of above frames sent is received by sender
lab2@lab2-OptiPlex-3090:~/Downloads$
```

Version 2: Client – Server Version

Server:

```
#include<stdio.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>
#include<stdlib.h>
#include<arpa/inet.h>
#include <unistd.h>
#define SIZE 4
int main()
{
int sfd,lfd,len,i,j,status;
char str[20],frame[20],temp[20],ack[20];
struct sockaddr_in saddr,caddr;
sfd=socket(AF_INET,SOCK_STREAM,0);
if(sfd<0)
perror("Error");
bzero(&saddr,sizeof(saddr));
saddr.sin_family=AF_INET;
saddr.sin_addr.s_addr=htonl(INADDR_ANY);
saddr.sin_port=htons(5465);
if(bind(sfd,(struct sockaddr*)&saddr,sizeof(saddr))<0)
perror("Bind Error");
listen(sfd,5);
len=sizeof(&caddr);
lfd=accept(sfd,(struct sockaddr*)&caddr,&len);
printf(" Enter the text : \n");
scanf("%s",str);
```

```
i=0;
while(i<strlen(str))
{
memset(frame,0,20);
strncpy(frame,str+i,SIZE);
printf(" Transmitting Frames. ");
len=strlen(frame);
for(j=0;j<len;j++)
{
printf("%d",i+j);
sprintf(temp,"%d",i+j);
strcat(frame,temp);
}
printf("\n");
write(lfd,frame,sizeof(frame));
read(lfd,ack,20);
sscanf(ack,"%d",&status);
if(status==-1)printf(" Transmission is successful. \n");
else
{
printf(" Received error in %d \n\n",status);
printf("\n\n Retransmitting Frame. ");
for(j=0;;)
{
frame[j]=str[j+status];
printf("%d",j+status);
j++;
if((j+status)%4==0)
break;
}
printf("\n");
frame[j]='\0';
len=strlen(frame);
for(j=0;j<len;j++)
{
sprintf(temp,"%d",j+status);
strcat(frame,temp);
}
write(lfd,frame,sizeof(frame));
}
i+=SIZE;
}
```

```
write(lfd,"exit",sizeof("exit"));
printf("Exiting\n");
sleep(2);
close(lfd);
close(sfd);
}
```

Output:

```
lab2@lab2-OptiPlex-3090:~/Downloads$ gcc SWP_S.c -o server
lab2@lab2-OptiPlex-3090:~/Downloads$ ./server
 Enter the text :
CNLABCSESRKR
 Transmitting Frames. 0123
 Transmission is successful.
 Transmitting Frames. 4567
 Transmission is successful.
 Transmitting Frames. 891011
 Received error in 9


 Retransmitting Frame. 91011
Exiting
lab2@lab2-OptiPlex-3090:~/Downloads$
```

Client:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include <unistd.h>
int main()
{  int sfd,lfd,len,choice;
char str[20],str1[20],err[20];
struct sockaddr_in saddr,caddr;
sfd=socket(AF_INET,SOCK_STREAM,0);
if(sfd<0)
perror("FdError");
bzero(&saddr,sizeof(saddr));
saddr.sin_family=AF_INET;
saddr.sin_addr.s_addr=INADDR_ANY;
saddr.sin_port=htons(5465);
connect(sfd,(struct sockaddr*)&saddr,sizeof(saddr));
for(;;)
{read(sfd,str,20);
if(!strcmp(str,"exit"))
{
```

```
printf("Exiting\n");
break;}
printf("\nReceived: %s\n1.Do u want to report an error(1-Yes 0-No):",str);
scanf("%d",&choice);
if(!choice)
write(sfd,"-1",sizeof("-1"));
else
{  printf("Enter the sequence no of the frame where error has occured:\n");
scanf("%s",err);
write(sfd,err,sizeof(err));
read(sfd,str,20);
printf("\nReceived the re-transmitted frames: %s\n",str);
} }  }
```

Output:

```
lab2@lab2-OptiPlex-3090:~/Downloads$ gcc SWP_C.c -o client
lab2@lab2-OptiPlex-3090:~/Downloads$ ./client

Received: CNLA0123
1.Do u want to report an error(1-Yes 0-No):0

Received: BCSE4567
1.Do u want to report an error(1-Yes 0-No):0

Received: SRKR891011
1.Do u want to report an error(1-Yes 0-No):1
Enter the sequence no of the frame where error has occured:
9

Received the re-transmitted frames: RKR91011
Exiting
lab2@lab2-OptiPlex-3090:~/Downloads$
```

Network Layer

- The Network Layer is the third layer of the OSI model.
- It handles the service requests from the transport layer and further forwards the service request to the data link layer.
- The network layer translates the logical addresses into physical addresses
- It determines the route from the source to the destination and manages the traffic problems such as switching, routing and controls the congestion of data packets.
- The main role of the network layer is to move the packets from sending host to the receiving host.

The main functions performed by the network layer are:

- **Routing:** When a packet reaches the router's input link, the router will move the packets to the router's output link. For example, a packet from S1 to R1 must be forwarded to the next router on the path to S2.
- **Logical Addressing:** The data link layer implements the physical addressing and network layer implements the logical addressing. Logical addressing is also used to distinguish between source and destination system. The network layer adds a header to the packet which includes the logical addresses of both the sender and the receiver.
- **Internetworking:** This is the main role of the network layer that it provides the logical connection between different types of networks.
- **Fragmentation:** The fragmentation is a process of breaking the packets into the smallest individual data units that travel through different networks.

Implementation of Address Resolution Protocol

Description:

Most of the computer programs/applications use logical address (IP address) to send/receive messages, however, the actual communication happens over the physical address (MAC address) i.e from layer 2 of the OSI model. So our mission is to get the destination MAC address which helps in communicating with other devices. This is where ARP comes into the picture, its functionality is to translate IP address to physical addresses.

The devices of the network peel the header of the data link layer from the protocol data unit (PDU) called frame and transfer the packet to the network layer (layer 3 of OSI) where the network ID of the packet is validated with the destination IP's network ID of the packet and if it's equal then it responds to the source with the MAC address of the destination, else the packet reaches the gateway of the network and broadcasts packet to the devices it is connected with and validates their network ID

The above process continues till the second last network device in the path reaches the destination where it gets validated and ARP, in turn, responds with the destination MAC address.
ARP: ARP stands for (Address Resolution Protocol). It is responsible to find the hardware address of a host from a known IP address. There are three basic ARP terms.

Program:

```
#include<sys/types.h>
#include<sys/socket.h>
#include<net/if_arp.h>
#include<sys/ioctl.h>
#include<stdio.h>
#include<string.h>
#include<unistd.h>
#include<math.h>
#include<complex.h>
#include<arpa/inet.h>
#include<netinet/in.h>
#include<netinet/if_ether.h>
#include<net/ethernet.h>
#include<stdlib.h>
int main(int argc,char *argv[])
{
 struct sockaddr_in sin={0};
 struct arpreq myarp={{0}};
 unsigned char *ptr;
 int sd;
 sin.sin_family=AF_INET;
 if(inet_aton(argv[1],&sin.sin_addr)==0)
 {
 printf("IP address Entered '%s' is not valid \n",argv[1]);
 exit(0);
 }
 memcpy(&myarp.arp_pa,&sin,sizeof(myarp.arp_pa));
 strcpy(myarp.arp_dev,"echo");
 sd=socket(AF_INET,SOCK_DGRAM,0);

 if(ioctl(sd,SIOCGARP,&myarp)==1)
 {
 printf("No Entry in ATP cache for '%s'\n",argv[1]);
```

```
 exit(0);
 }
 ptr=&myarp.arp_pa.sa_data[0];
 printf("MAC Address for '%s' : ",argv[1]);
 printf("%x:%x:%x:%x:%x:%x\n",*ptr,*(ptr+1),*(ptr+2),*(ptr+3),*(ptr+4),*(ptr+5));
 return 0;
}
```

Output:

```
lab5@lab5-OptiPlex-3020:~/Desktop$ gcc arp.c
lab5@lab5-OptiPlex-3020:~/Desktop$ ./a.out 172.19.27.44
MAC Address for '172.19.27.44' : 0:0:ac:13:1b:2c
lab5@lab5-OptiPlex-3020:~/Desktop$
```

Implementation of distance vector routing algorithm.

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).

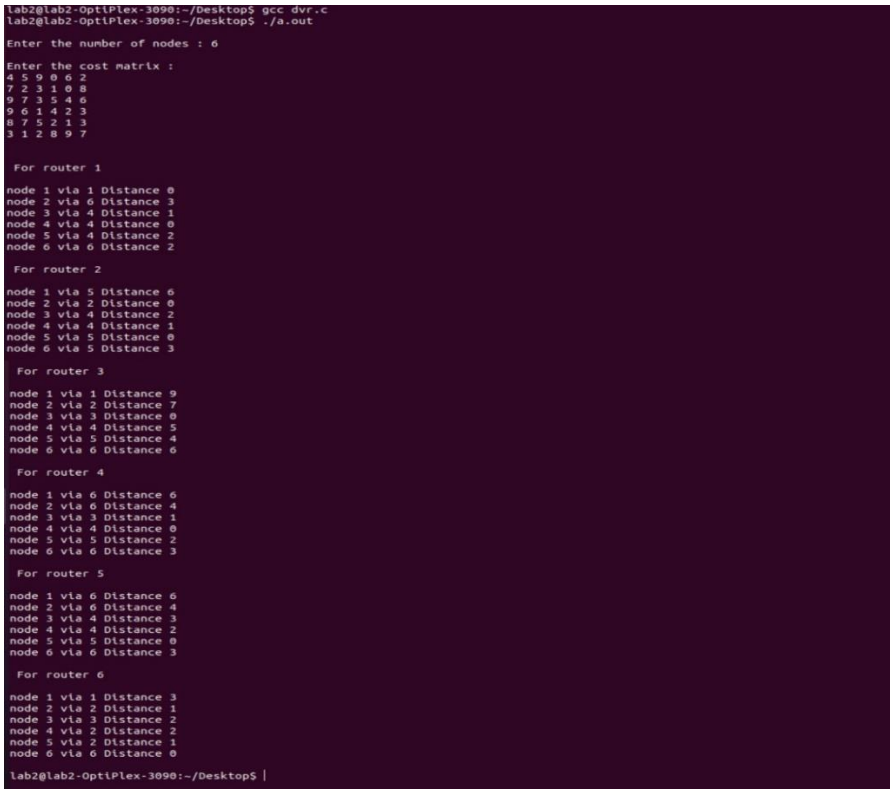
Bellman Ford – Each router maintains a Distance Vector table containing the distance between itself and ALL possible destination nodes. Distances, based on a chosen metric, are computed using information from the neighbour's distance vectors.

Program:

```
#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");      }
```

Output:



Implementation of Dijkstra's algorithm.

Description:

Dijkstra's algorithm allows us to find the shortest path between any two vertices of a graph.It differs from the minimum spanning tree because the shortest distance between two vertices might not include all the vertices of the graph.

Djikstra used this property in the opposite direction i.e we overestimate the distance of each vertex from the starting vertex. Then we visit each node and its neighbors to find the shortest subpath to those neighbors.

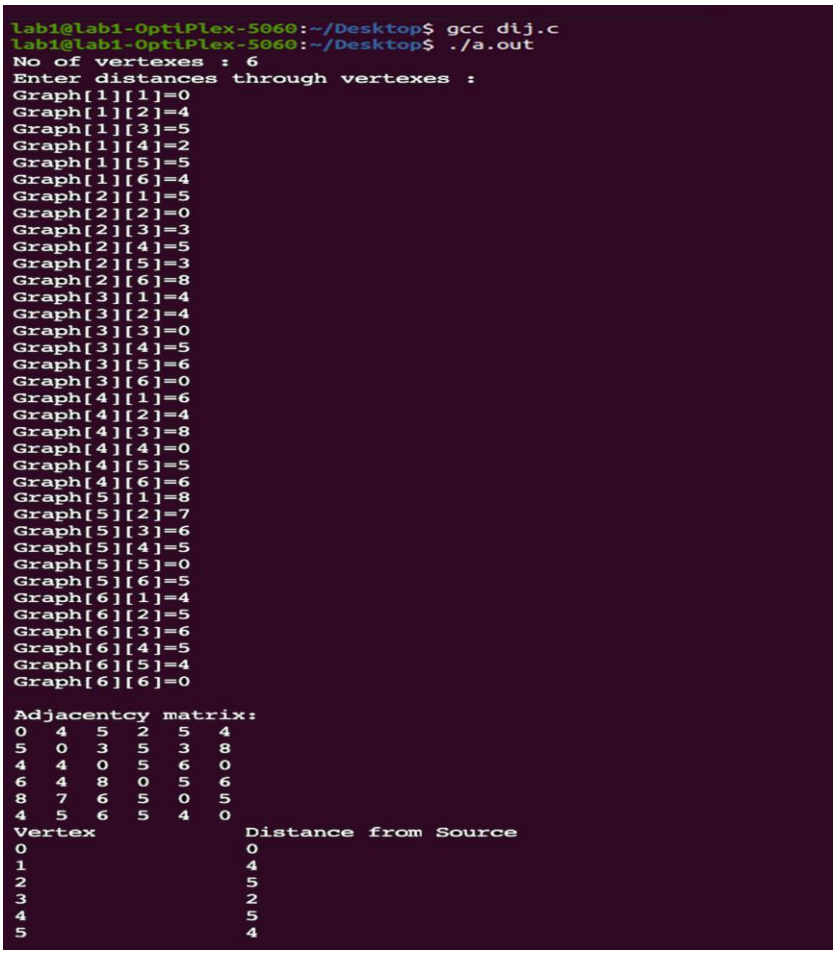
The algorithm uses a greedy approach in the sense that we find the next best solution hoping that the end result is the best solution for the whole problem.

Program:

```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
int V=20;
int minDistance(int dist[], bool sptSet[])
{
    int min = INT_MAX, min_index;
    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;
    return min_index;}
void printSolution(int dist[])
{    printf("Vertex \t\t Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%d \t\t %d\n", i, dist[i]);
}
void dijkstra(int graph[V][V], int src)
{    int dist[V]; // The output array. dist[i] will hold the
    bool sptSet[V]; // sptSet[i] will be true if vertex i is
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;
    dist[src] = 0;
    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, sptSet);
        sptSet[u] = true;
        for (int v = 0; v < V; v++)
            if (!sptSet[v] && graph[u][v]
                && dist[u] != INT_MAX
                && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];    }
    printSolution(dist);        }
```

```
int main()
{
int graph[V][V],i,j;
     printf("No of vertexes : ");
scanf("%d",&V);
printf("Enter distances through vertexes : \n");
for(i=0;i<V;i++)
{for(j=0;j<V;j++)
{  printf("Graph[%d][%d]=",i+1,j+1);
scanf("%d",&graph[i][j]);
}  }
printf("\nAdjacentcy matrix:\n");
for(i=0;i<V;i++)
{  for(j=0;j<V;j++)
{   printf("%d ",graph[i][j]);
}     printf("\n");
}
dijkstra(graph, 0);
    return 0;     }
```

Output:



Implementation of Open Shortest Path First (OSPF) routing protocol

Description:

The OSPF stands for Open Shortest Path First. It is a widely used and supported routing protocol. It is an intradomain protocol, which means that it is used within an area or a network. It is an interior gateway protocol that has been designed within a single autonomous system. It is based on a link-state routing algorithm in which each router contains the information of every domain, and based on this information, it determines the shortest path. The goal of routing is to learn routes. The OSPF achieves by learning about every router and subnet within the entire network. Every router contains the same information about the network. The way the router learns this information by sending LSA (Link State Advertisements). These LSAs contain information about every router, subnet, and other networking information. Once the LSAs have been flooded, the OSPF stores the information in a link-state database known as LSDB. The main goal is to have the same information about every router in an LSDBs.

Program:

```
#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:");
for(i=0;i<count;i++)
{    for(j=0;j<count;j++)
{
printf("%d->%d:",i,j);
scanf("%d",&cost_matrix[i][j]);
if(cost_matrix[i][j]<0)cost_matrix[i][j]=1000;
}}
printf("\nEnter 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("%d==>%d:Path taken:%d",src_router,i,i);
w=i;
while(w!=src_router)
{    printf("<--%d",last[w]);w=last[w];
}
printf("\nShortest path cost:%d",dist[i]);
}      }
```

Output:

```
lab2@lab2-OptiPlex-3090:~/Downloads$ gcc ospf.c
lab2@lab2-OptiPlex-3090:~/Downloads$ ./a.out

 Enter the no of routers: 6

 Enter the cost matrix values:0->0:4
0->1:2
0->2:5
0->3:4
0->4:6
0->5:2
1->0:2
1->1:3
1->2:1
1->3:2
1->4:45
1->5:52
2->0:0
2->1:1
2->2:2
2->3:3
2->4:4
2->5:2
3->0:5
3->1:6
3->2:2
3->3:3
3->4:2
3->5:1
4->0:2
4->1:2
4->2:3
4->3:25
4->4:5
4->5:2
5->0:3
5->1:2
5->2:0
5->3:1
5->4:2
5->5:4

Enter the source router: 2
2==>0:Path taken:0<--2
Shortest path cost:02==>1:Path taken:1<--2
Shortest path cost:12==>2:Path taken:2
Shortest path cost:22==>3:Path taken:3<--2
Shortest path cost:32==>4:Path taken:4<--2
Shortest path cost:42==>5:Path taken:5<--2
lab2@lab2-OptiPlex-3090:~/Downloads$
```

Implementation of broadcast tree

Description:

Broadcasting in computer network is a group communication, where a sender sends data to receivers simultaneously. This is an all − to − all communication model where each sending device transmits data to all other devices in the network domain.

The ways of operation of broadcasting may be

1. A high level operation in a program, like broadcasting in Message Passing Interface.
2. A low level networking operation, like broadcasting on Ethernet.

Program:

```
#include<stdio.h>
int p,q,u,v,n,i,j;;
int min=99,mincost=0;
int t[50][2],parent[50],edge[50][50];
void Sunion(int l,int m)
{
parent[l]=m;
}

int find(int k)
{
if(parent[k]>0)
k=parent[k];
return k;
}
int main()
{
printf("\n Enter the number of nodes: ");
scanf("\t%d",&n);
printf("Enter Matrix:\n");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
scanf("%d",&edge[i][j]);
}
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
if(edge[i][j]!=99)
if(min>edge[i][j])
```

```
{
min=edge[i][j];
u=i;
v=j;
}
p=find(u);
q=find(v);
if(p!=q)
{
t[i][0]=u;
t[i][1]=v;
mincost=mincost+edge[u][v];
Sunion(p,q);
}
else
{
t[i][0]=-1;
t[i][1]=-1;
}
min=99;
}
printf("Minimum cost is %d\nMinimum spanning tree is\n" ,mincost);
for(i=0;i<n;i++)
if(t[i][0]!=-1 && t[i][1]!=-1)
{
printf("%c %c %d", 65+t[i][0],65+t[i][1],edge[t[i][0]][t[i][1]]);
printf("\n");
}           }
```

Output:

```
lab5@lab5-OptiPlex-3020:~/Desktop$ gcc broadtree.c
lab5@lab5-OptiPlex-3020:~/Desktop$ ./a.out

 Enter the number of nodes: 6
Enter Matrix:
5 4 7 8 9 2
4 4 5 1 2 4
0 3 2 4 5 2
3 5 4 2 2 1
7 4 5 1 4 5
2 0 1 4 3 3
Minimum cost is 5
Minimum spanning tree is
A F 2
B D 1
C A 0
D F 1
E D 1
F B 0
lab5@lab5-OptiPlex-3020:~/Desktop$
```

Transport layer

The transport layer is the fourth layer in the open systems interconnection (OSI) network model. All modules and procedures pertaining to transportation of data or data stream are categorized into this layer. As all other layers, this layer communicates with its peer Transport layer of the remote host.

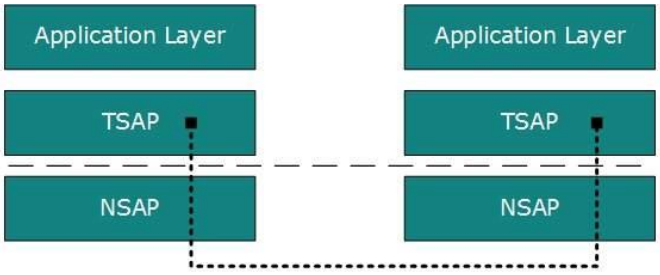
Transport layer offers peer-to-peer and end-to-end connection between two processes on remote hosts. Transport layer takes data from upper layer (i.e. Application layer) and then breaks it into smaller size segments, numbers each byte, and hands over to lower layer (Network Layer) for delivery.

Functions

- This Layer is the first one which breaks the information data, supplied by Application layer in to smaller units called segments. It numbers every byte in the segment and maintains their accounting.
- This layer ensures that data must be received in the same sequence in which it was sent.
- This layer provides end-to-end delivery of data between hosts which may or may not belong to the same subnet.
- All server processes intend to communicate over the network are equipped with well-known Transport Service Access Points (TSAPs) also known as port numbers.

End-to-End Communication

A process on one host identifies its peer host on remote network by means of TSAPs, also known as Port numbers. TSAPs are very well defined and a process which is trying to communicate with its peer knows this in advance.



For example, when a DHCP client wants to communicate with remote DHCP server, it always requests on port number 67. When a DNS client wants to communicate with remote DNS server, it always requests on port number 53 (UDP).

The two main Transport layer protocols are:

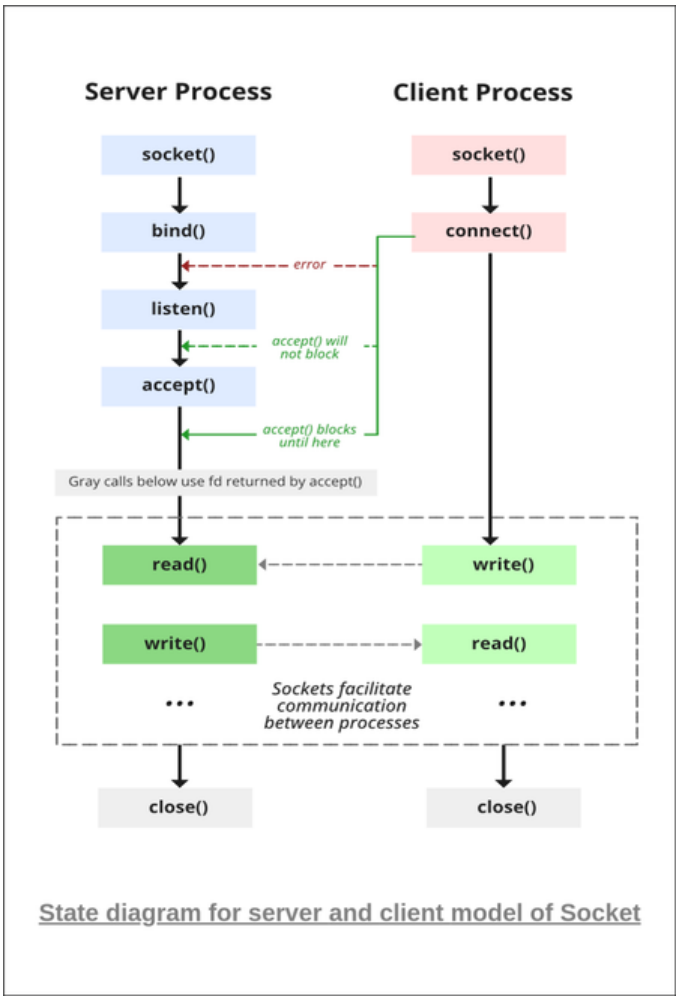
- **Transmission Control Protocol**
  It provides reliable communication between two hosts.
- **User Datagram Protocol**
  It provides unreliable communication between two hosts.

Socket programming:

Socket and Port are the terms used in Transport Layer. A port is a logical construct assigned to network processes so that they can be identified within the system. A socket is a combination of port and IP address. An incoming packet has a port number which is used to identify the process that needs to consume the packet.
The lowest numbered 1024 port numbers are used for the most used services. These ports are called the well-known ports. Higher-numbered ports are available for general use by applications and are known as ephemeral ports.

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

State diagram for server and client model



State diagram for server and client model of Socket

Stages for server

1. Socket creation:

int sockfd = socket(domain, type, protocol)

- **sockfd:** socket descriptor, an integer (like a file-handle)
- **domain:** integer, specifies communication domain. We use AF_ LOCAL as defined in the POSIX standard for communication between processes on the same host. For communicating between processes on different hosts connected by IPV4, we use AF_INET and AF_I NET 6 for processes connected by IPV6.
- **type:** communication type
  SOCK_STREAM: TCP(reliable, connection oriented)
  SOCK_DGRAM: UDP(unreliable, connectionless)
- **protocol:** Protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on protocol field in the IP header of a packet.(man protocols for more details)

2. Setsockopt:

This helps in manipulating options for the socket referred by the file descriptor sockfd. This is completely optional, but it helps in reuse of address and port. Prevents error such as: "address already in use".

*int setsockopt(int sockfd, int level, int optname, const void *optval, socklen_t optlen);*

3. Bind:

*int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);*

After the creation of the socket, the bind function binds the socket to the address and port number specified in addr(custom data structure). In the example code, we bind the server to the localhost, hence we use INADDR_ANY to specify the IP address.

4. Listen:

int listen(int sockfd, int backlog);

It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog, defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.

5. Accept:

*int new_socket= accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);*

It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket. At this point, the connection is established between client and server, and they are ready to transfer data.

Stages for Client

- **Socket connection:** Exactly same as that of server's socket creation

- **Connect:** The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server's address and port is specified in addr.

*int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);*

Creating client-server application using TCP Socket

Description:

The InterSystems IRIS Transmission Control Protocol (TCP) binding. Establishes a two-way connection between a server and a single client. Provides reliable byte stream transmission of data with error checking and correction, and message acknowledgement.

The TCP binding connects InterSystems IRIS to a widespread networking standard so that basic features of the underlying network protocol are available to InterSystems IRIS users through I/O commands.

The TCP/IP protocol allows systems to communicate even if they use different types of network hardware. For example, TCP, through an Internet connection, transmits messages between a system using Ethernet and another system using Token Ring. TCP controls the accuracy of data transmission. IP, or Internet Protocol, performs the actual data transfer between different systems on the network or Internet.

Using TCP binding, you can create both client and server portions of client-server systems. In the client-server type of distributed database system, users on one or more client systems can process information stored in a database on another system, called the server.

Program:

TCP Server:

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#include <sys/types.h>
#define PORT 8080
int main(int argc, char const* argv[])
{
int server_fd, new_socket, valread;
struct sockaddr_in address;
int opt = 1;
int addrlen = sizeof(address);
```

```
char buffer[1024] = { 0 };
char* hello = "Hello from Server";

// Creating socket file descriptor
if ((server_fd = socket(AF_INET, SOCK_STREAM, 0))== 0) {
perror("socket failed");
exit(EXIT_FAILURE);
}
// Forcefully attaching socket to the port 8080
if (setsockopt(server_fd, SOL_SOCKET,SO_REUSEADDR | SO_REUSEPORT,
&opt,sizeof(opt)))
{
perror("setsockopt");
exit(EXIT_FAILURE);
}
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(PORT);

// Forcefully attaching socket to the port 8080
if (bind(server_fd, (struct sockaddr*)&address,sizeof(address))< 0) {
perror("bind failed");
exit(EXIT_FAILURE);
}
if (listen(server_fd, 5) < 0) {
perror("listen");
exit(EXIT_FAILURE);
}
if ((new_socket= accept(server_fd, (struct
sockaddr*)&address,(socklen_t*)&addrlen))< 0)
{
perror("accept");
exit(EXIT_FAILURE);
}
valread = read(new_socket, buffer, 1024);
printf("%s\n", buffer);

send(new_socket, hello, strlen(hello), 0);
printf("Hello message sent\n");
// closing the connected socket
close(new_socket);
// closing the listening socket
```

```
shutdown(server_fd, SHUT_RDWR);
return 0;
}
```

Output:

```
lab5@lab5-OptiPlex-3020:~/Desktop$ gcc tcpserver.c -o server
lab5@lab5-OptiPlex-3020:~/Desktop$ ./server
hello from client
Hello message sent
lab5@lab5-OptiPlex-3020:~/Desktop$
```

TCP Client:

Program:

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define PORT 8080
int main(int argc, char const* argv[])
{ int sock = 0, valread, client_fd;
struct sockaddr_in serv_addr;
char* hello = "hello from client";
char buffer[1024] = { 0 };
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
printf("\n Socket creation error \n");
return -1;
}

serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons(PORT);

if ((client_fd= connect(sock, (struct sockaddr*)&serv_addr,sizeof(serv_addr)))< 0)
{
printf("\nConnection Failed \n");
return -1;
}
send(sock, hello, strlen(hello), 0);
```

```
printf("Hello message sent\n");
valread = read(sock, buffer, 1024);
printf("%s\n", buffer);

// closing the connected socket
close(client_fd);
return 0;
}
```

Output:

```
lab5@lab5-OptiPlex-3020:~/Desktop$ gcc tcpclient.c -o client
lab5@lab5-OptiPlex-3020:~/Desktop$ ./client
Hello message sent
Hello from Server
lab5@lab5-OptiPlex-3020:~/Desktop$
```

Creating client-server application using UDP Socket

Description:

UDP is a simple transport-layer protocol. The application writes a message to a UDP socket, which is then encapsulated in a UDP datagram, which is further encapsulated in an IP datagram, which is sent to the destination. There is no guarantee that a UDP will reach the destination, that the order of the datagrams will be preserved across the network or that datagrams arrive only once.

The problem of UDP is its lack of reliability: if a datagram reaches its destination but the checksum detects an error, or if the datagram is dropped in the network, it is not automatically retransmitted. Each UDP datagram is characterized by a length. The length of a datagram is passed to the receiving application along with the data. No connection is established between the client and the server, and, for this reason, we say that UDP provides a *connection-less service*.

The steps of establishing a UDP socket communication on the client side are as follows:

- Create a socket using the `socket()` function;
- Send and receive data by means of the `recvfrom()` and `sendto()` functions.

The steps of establishing a UDP socket communication on the server side are as follows:

- Create a socket with the `socket()` function;

- Bind the socket to an address using the `bind()` function;
- Send and receive data by means of `recvfrom()` and `sendto()`.

Program:

UDP Server:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include<stdlib.h>
int main(int argc, char *argv[]) {
 int sd, rc, n, cliLen;
 struct sockaddr_in cliAddr, servAddr;
 char msg[100];
 /* socket creation */
 sd=socket(AF_INET, SOCK_DGRAM, 0);
 if(sd<0) {
 printf("%s: cannot open socket \n",argv[0]);
 exit(1);
 }
 /* bind local server port */
 servAddr.sin_family = AF_INET;
 servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
 servAddr.sin_port = htons(2525);
 rc = bind (sd, (struct sockaddr *) &servAddr,sizeof(servAddr));
 if(rc<0) {
 printf("%s: cannot bind port number 2525 \n",
 argv[0]);
 exit(1);
 }
 printf("%s: waiting for data on port UDP 2525\n",
 argv[0]);
 /* server infinite loop */
 while(1) {

 /* init buffer */
```

```
 memset(msg,0x0,100);
 /* receive message */
 cliLen = sizeof(cliAddr);
 n = recvfrom(sd, msg, 20, 0,
(struct sockaddr *) &cliAddr, &cliLen);
 if(n<0) {
 printf("%s: cannot receive data \n",argv[0]);
 continue;
 }
 /* print received message */
 printf("%s: from %s:UDP%u : %s \n",
 argv[0],inet_ntoa(cliAddr.sin_addr),
 ntohs(cliAddr.sin_port),msg);
 }/* end of server infinite loop */
return 0;
}
```

```
sindhu@JARVIS:~/cnlab/UDP$ ./server
./server: waiting for data on port UDP 2525
./server: from 172.28.217.71:UDP48316 : Hello_from_client
```

UDP Client:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/time.h>
#include<stdlib.h>
int main(int argc, char *argv[]) {
 int sd, rc, i;
 struct sockaddr_in cliAddr, remoteServAddr;
 struct hostent *h;
 if(argc<3) {
 printf("usage : %s <server> <data1> ... <dataN> \n", argv[0]);
 exit(1);
 }
 h = gethostbyname(argv[1]);
 if(h==NULL) {
```

```
 printf("%s: unknown host '%s' \n", argv[0], argv[1]);
 exit(1);
 }
 printf("%s: sending data to '%s' (IP : %s) \n", argv[0], h->h_name,
inet_ntoa(*(struct in_addr *)h->h_addr_list[0]));
 remoteServAddr.sin_family = h->h_addrtype;
 memcpy((char *) &remoteServAddr.sin_addr.s_addr,
h->h_addr_list[0], h->h_length);
 remoteServAddr.sin_port = htons(2525);
 sd = socket(AF_INET,SOCK_DGRAM,0);
 if(sd<0) {
 printf("%s: cannot open socket \n",argv[0]);
 exit(1);  }
 cliAddr.sin_family = AF_INET;
 cliAddr.sin_addr.s_addr = htonl(INADDR_ANY);
 cliAddr.sin_port = htons(0);
 rc = bind(sd, (struct sockaddr *) &cliAddr, sizeof(cliAddr));
 if(rc<0) {
 printf("%s: cannot bind port\n", argv[0]);
 exit(1);  }
 /* send data */
 for(i=2;i<argc;i++) {
 rc = sendto(sd, argv[i], strlen(argv[i])+1, 0,
(struct sockaddr *) &remoteServAddr,
sizeof(remoteServAddr));
 if(rc<0) {
 printf("%s: cannot send data %d \n",argv[0],i-1);
 close(sd);
 exit(1);
 }
 }
 return 1;
}
```

```
sindhu@JARVIS:~/cnlab/UDP$ ./client 172.28.217.71 Hello_from_client
./client: sending data to '172.28.217.71' (IP : 172.28.217.71)
sindhu@JARVIS:~/cnlab/UDP$
```

Implementation of Domain name system (finding IP address for given domain name)

Description:

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

- DNS stands for Domain Name System
- DNS is a directory service that provides a mapping between the name of a host on the network and its numerical address.
- DNS is required for the functioning of the internet.
- Each Node in a tree has a domain name and a full domain name is a sequence of symbols specified by dots.

Program:

```
#include<stdlib.h>
#include<errno.h>
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<unistd.h>
#include<arpa/inet.h>
#include<netdb.h>
int main(int argc,char*argv[1])
{
 struct hostent*hen;
 if(argc!=2)
  {
    fprintf(stderr,"enter host name\n");
  }
  hen=gethostbyname(argv[1]);
  if(hen==NULL)
  {
    fprintf(stderr,"host not found");
  }
  printf("Host name: %s \n",hen->h_name);
  printf("IP address: %s \n",inet_ntoa(*((struct in_addr *)hen->h_addr)));
}
```

Output:

```
lab5@lab5-OptiPlex-3020:~/Desktop$ gcc dns.c
lab5@lab5-OptiPlex-3020:~/Desktop$ ./a.out www.srkrec.ac.in
Host name: srkrec.ac.in
IP address: 43.250.40.60
lab5@lab5-OptiPlex-3020:~/Desktop$
```