COMPUTER NETWORKS
LAB WORK BOOK
(PCC- CS692)

INSTITUTE OF ENGINEERING AND
MANAGEMENT SALT LAKE, KOLKATA

Name of Student:Sreya Adhikary
Semester: 6th Semester
Roll No.: 43
Stream: CSE AIML
Enrollment No.: 12020002016055
Year: 3rd Year

INDEX

Experiment 1:

Familiarization with o Networking cables

1. CAT5, UTP
2. Connectors RJ45, T-connector
3. Hubs, Switches

Theory:

Category 5 cable (Cat 5) is a twisted pair cable for computer networks. The cable standard provides performance of up to 100 MHz and is suitable for most varieties of Ethernet over twisted pair up to 2.5GBASE-T but more commonly runs at 1000BASE-T (Gigabit Ethernet) speeds. Cat 5 is also used to carry other signals such as telephone and video. This cable is commonly connected using punch-down blocks and modular connectors.

Unshielded Twisted Pair (UTP) cable is most popular cable around the world. UTP cable is used not only for networking but also for the **traditional telephone** (**UTP-Cat 1**). There are **seven different types of UTP categories** and, depending on what you want to achieve, you would need the appropriate type of cable. **UTP-CAT5e** is the most popular UTP cable.

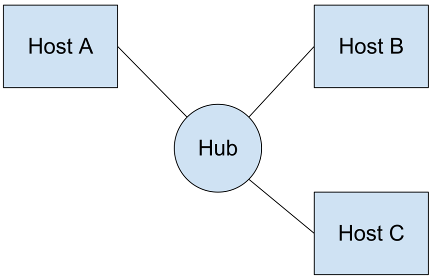
Connectors RJ45 are a type of connector used for Ethernet networking. The term "RJ45" refers to the physical interface used to connect network cables to networking devices such as routers, switches, and network interface cards (NICs). RJ45 connectors have eight pins and are commonly used with twisted pair cables, which are often referred to as Ethernet cables. These cables are used to transmit data over a network using the Ethernet protocol, and they are widely used in both home and business networking environments.

T-connector is a type of connector that can be used with coaxial cables to split the signal into two separate lines or to connect a single cable to multiple devices. However, it's important to note that T-connectors are not commonly used in modern Ethernet networks, as they have been largely replaced by switches.

A hub is a device that connects multiple devices together in a local area network (LAN). Hubs are considered as the most basic type of networking device, which works at the physical layer of the OSI model. A hub contains multiple ports, which allows multiple devices, such as computers, printers, and servers, to be connected to a single network segment.

Switches operate at the data link layer of the OSI model, which is layer 2, and they are often used in Ethernet networks. A switch contains multiple ports, which allows multiple devices, such as computers, printers, and servers, to be connected to a single network segment.

Diagram:



Code:

Output:

<u>Experiment 2:</u> TCP Socket Programming Multicast & Broadcast Sockets

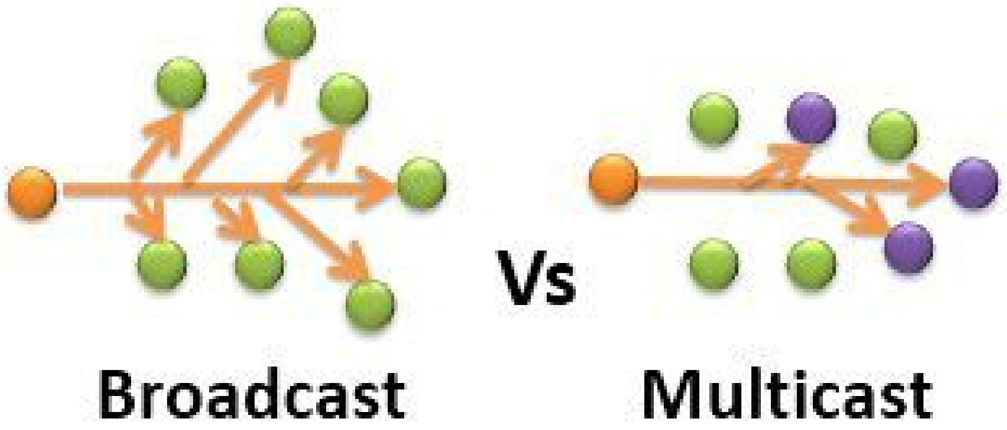
Theory:

TCP socket programming can be used to implement both multicast and broadcast sockets:

Multicast sockets are used to send data to a group of devices that have joined a specific multicast group. The sender sends the data to the multicast group, and the multicast router in the network forwards the data to all devices that have joined the group. In TCP socket programming, the multicast feature is implemented using the Internet Group Management Protocol (IGMP) to manage the multicast groups.

Broadcast sockets, on the other hand, are used to send data to all devices on a network segment. The sender sends the data to the broadcast address, which is the highest address in the network segment, and the data is then delivered to all devices on that segment. In TCP socket programming, the broadcast feature is implemented using the INADDR_BROADCAST IP address.

Diagram:



Code:

Server Side Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

int main(){

  char *ip = "127.0.0.1";
  int port = 5566;

  int server_sock, client_sock;
  struct sockaddr_in server_addr, client_addr;
  socklen_t addr_size;
  char buffer[1024];
  int n;

  server_sock = socket(AF_INET, SOCK_STREAM, 0);
  if (server_sock < 0){
    perror("[-]Socket error");
    exit(1);
  }
  printf("[+]TCP server socket created.\n");

  memset(&server_addr, '\0', sizeof(server_addr));
  server_addr.sin_family = AF_INET;
  server_addr.sin_port = port;
  server_addr.sin_addr.s_addr = inet_addr(ip);

  n = bind(server_sock, (struct sockaddr*)&server_addr, sizeof(server_addr));
  if (n < 0){
    perror("[-]Bind error");
    exit(1);
  }
  printf("[+]Bind to the port number: %d\n", port);

  listen(server_sock, 5);
  printf("Listening...\n");

  while(1){
    addr_size = sizeof(client_addr);
    client_sock = accept(server_sock, (struct sockaddr*)&client_addr, &addr_size);
    printf("[+]Client connected.\n");

    bzero(buffer, 1024);
    recv(client_sock, buffer, sizeof(buffer), 0);
    printf("Client: %s\n", buffer);

    bzero(buffer, 1024);
    strcpy(buffer, "HI, THIS IS SERVER. HAVE A NICE DAY!!!");
    printf("Server: %s\n", buffer);
    send(client_sock, buffer, strlen(buffer), 0);

    close(client_sock);
    printf("[+]Client disconnected.\n\n");

  }
  return 0;
}
```

Output:

```
[+]TCP server socket created.
[+]Bind to the port number: 5566
Listening…
```

Client Side Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

int main(){

  char *ip = "127.0.0.1";
  int port = 5566;
  int sock;
  struct sockaddr_in addr;
  socklen_t addr_size;
  char buffer[1024];
  int n;

  sock = socket(AF_INET, SOCK_STREAM, 0);
  if (sock < 0){
    perror("[-]Socket error");
    exit(1);
  }
  printf("[+]TCP server socket created.\n");

  memset(&addr, '\0', sizeof(addr));
  addr.sin_family = AF_INET;
  addr.sin_port = port;
  addr.sin_addr.s_addr = inet_addr(ip);

  connect(sock, (struct sockaddr*)&addr, sizeof(addr));
  printf("Connected to the server.\n");

  bzero(buffer, 1024);
  strcpy(buffer, "HELLO, THIS IS CLIENT.");
  printf("Client: %s\n", buffer);
  send(sock, buffer, strlen(buffer), 0);

  bzero(buffer, 1024);
  recv(sock, buffer, sizeof(buffer), 0);
  printf("Server: %s\n", buffer);

  close(sock);
  printf("Disconnected from the server.\n");

  return 0;
}
```

Output:

```
[+]TCP server socket created.
Connected to the server.
Client: HELLO, THIS IS CLIENT.
Server:
Disconnected from the server.
```

Experiment 3: UDP Socket Programming Multicast & Broadcast Sockets

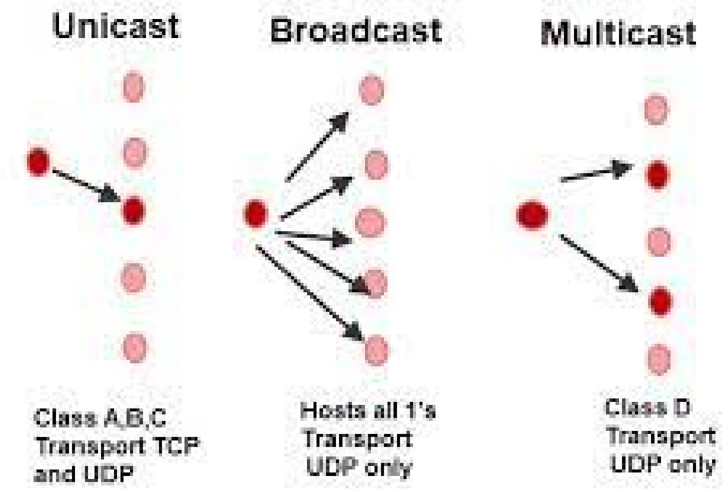
Theory:

UDP socket programming can also be used to implement multicast and broadcast sockets for sending data to multiple recipients in a network.

Multicast sockets in UDP are similar to TCP in that the sender sends data to a multicast group, and the data is then forwarded to all devices that have joined the group. The main difference is that in UDP, there is no connection between the sender and the receiver, and the data is sent as datagrams, without any guarantee of delivery or order. Multicast in UDP is implemented using the same Internet Group Management Protocol (IGMP) as in TCP.

Broadcast sockets in UDP work in a similar way to TCP. The sender sends data to the broadcast address, which is the highest address in the network segment, and the data is then delivered to all devices on that segment. In UDP, the broadcast feature is implemented using the INADDR_BROADCAST IP address, just like in TCP.

Diagram:



Unicast,Broadcast and Multicast IP Addressing

Code:

***Server Code*:**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
```

```
int main(int argc, char **argv){

  if (argc != 2) {
    printf("Usage: %s <port>\n", argv[0]);
    exit(0);
  }

  char *ip = "127.0.0.1";
  int port = atoi(argv[1]);

  int sockfd;
  struct sockaddr_in server_addr, client_addr;
  char buffer[1024];
  socklen_t addr_size;
  int n;

  sockfd = socket(AF_INET, SOCK_DGRAM, 0);
  if (sockfd < 0) {
    perror("[-]socket error");
    exit(1);
  }

  memset(&server_addr, '\0', sizeof(server_addr));
  server_addr.sin_family = AF_INET;
  server_addr.sin_port = htons(port);
  server_addr.sin_addr.s_addr = inet_addr(ip);

  n = bind(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr));
  if (n < 0){
    perror("[-]bind error");
    exit(1);
  }

  bzero(buffer, 1024);
  addr_size = sizeof(client_addr);
  recvfrom(sockfd, buffer, 1024, 0, (struct sockaddr*)&client_addr, &addr_size);
  printf("[+]Data recv: %s\n", buffer);

  bzero(buffer, 1024);
  strcpy(buffer, "Welcome to the UDP Server.");
  sendto(sockfd, buffer, 1024, 0, (struct sockaddr*)&client_addr, sizeof(client_addr));
  printf("[+]Data send: %s\n", buffer);

  return 0;
}
```

Client Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int main(int argc, char **argv){

  if (argc != 2) {
    printf("Usage: %s <port>\n", argv[0]);
    exit(0);
  }

  char *ip = "127.0.0.1";
  int port = atoi(argv[1]);

  int sockfd;
  struct sockaddr_in addr;
  char buffer[1024];
  socklen_t addr_size;

  sockfd = socket(AF_INET, SOCK_DGRAM, 0);
  memset(&addr, '\0', sizeof(addr));
  addr.sin_family = AF_INET;
  addr.sin_port = htons(port);
  addr.sin_addr.s_addr = inet_addr(ip);

  bzero(buffer, 1024);
  strcpy(buffer, "Hello World!");
  sendto(sockfd, buffer, 1024, 0, (struct sockaddr*)&addr, sizeof(addr));
  printf("[+]Data send: %s\n", buffer);

  bzero(buffer, 1024);
  addr_size = sizeof(addr);
  recvfrom(sockfd, buffer, 1024, 0, (struct sockaddr*)&addr, &addr_size);
  printf("[+]Data recv: %s\n", buffer);

  return 0;
}
```

<u>Experiment 4:</u>

Data Link Layer Flow Control Mechanism

1. Stop & Wait,
2. Sliding Window

Theory:

The data link layer is responsible for reliable transmission of data between two directly connected nodes over a communication link. Flow control is a mechanism used by the data link layer to control the rate of data transmission between the sender and the receiver to prevent the receiver from being overwhelmed with data.

There are two main flow control mechanisms used in the data link layer: stop-and-wait and sliding window.

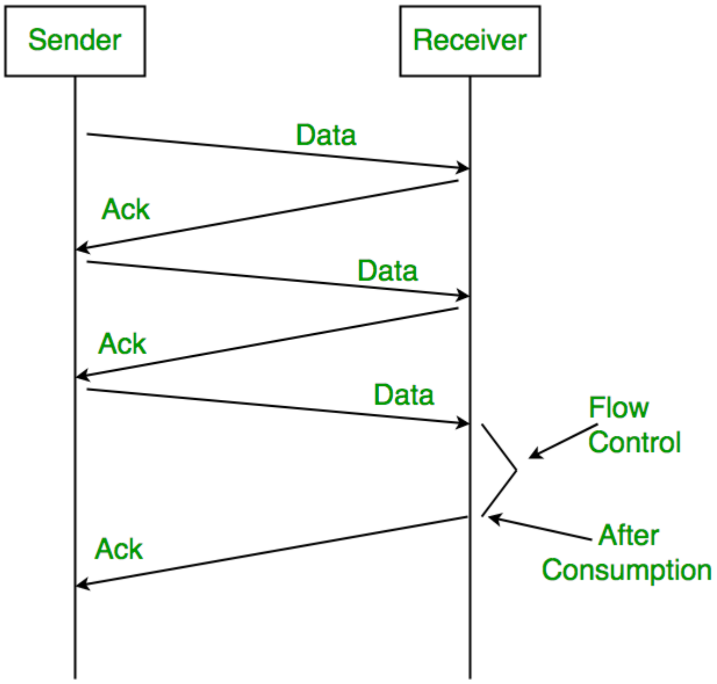
Stop-and-Wait Flow Control:

In stop-and-wait flow control, the sender sends one packet at a time and waits for an acknowledgement from the receiver before sending the next packet. Once the sender receives an acknowledgement, it sends the next packet. This process continues until all packets have been sent.

Sliding Window Flow Control:

In sliding window flow control, the sender can transmit multiple packets without waiting for an acknowledgement from the receiver for each packet. The sender sends a block of packets, called a window, and waits for an acknowledgement from the receiver for the entire window before sending the next window.

Diagram:



Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int k,time,win=2,i2=0,frame=0,a[20],b[20],i,j,s,r,ack,c,d;
int send(int,int);
int receive();
int checsum(int *);
main()
{
int i1=0,j1=0,c1;
printf("Enter the frame size\n");
scanf("%d",&frame);
printf("Enter the window size\n");
scanf("%d",&win);
j1=win;
for(i=0;i<frame;i++)
{
a[i]=rand();
}
k=1;
while(i1<frame)
{
if((frame-i1)<win)
j1=frame-i1;
printf("\n\ntransmit the window no %d\n\n",k);
c1=send(i1,i1+j1);
ack=receive(i1,i1+j1,c1);
if (ack!=0)
{printf("\n\n1.Selective window\n");
printf("2.Go back N\n");
scanf("%d",&ack);
switch(ack)
{
case 1:
printf("\n\n\t Selective window \t\nEnter the faulty frame no\n");
scanf("%d",&i2);
printf("\n\n Retransmit the frame %d \n",i2);
send(i2,i2+1);
break;
case 2:
printf("\n\n\t Go back n\t\n\n");
printf("\nRetransmit the frames from %d to %d\n",i1,i1+j1);
send(i1,i1+j1);
break;
}
}
 i1=i1+win;
 k++;
 }
 }
 int send(c,d)
 {
 int t1;
 for(i=c;i<d;i++)
 {
 b[i]=a[i];
 printf("frame %d is sent\n",i);
 }
 s=checsum(&a[c]);
 return(s); }
 int receive(c,d,c2)
 int c2;
 {
 r=checsum(&b[c]);
 if(c2==r)
 {
 return(0);
 }
else
return(1);
}
```

```
int checsum(int *c)
{
int sum=0;
for(i=0;i<win;i++)
sum=sum^(*c);
return sum;
}
```

Output:

```
Enter the frame size
50
Enter the window size
5
transmit the window no 1

frame 0 is sent
frame 1 is sent
frame 2 is sent
frame 3 is sent
frame 4 is sent


transmit the window no 2

frame 5 is sent
frame 6 is sent
frame 7 is sent
frame 8 is sent
frame 9 is sent
transmit the window no 3

frame 10 is sent
frame 11 is sent
frame 12 is sent
frame 13 is sent
frame 14 is sent


transmit the window no 4

frame 15 is sent
frame 16 is sent
frame 17 is sent
frame 18 is sent
frame 19 is sent
```

transmit the window no 5

frame 35005211 is sent

1.Selective window
2.Go back N

Experiment 5:

Data Link Layer Error Control Mechanism

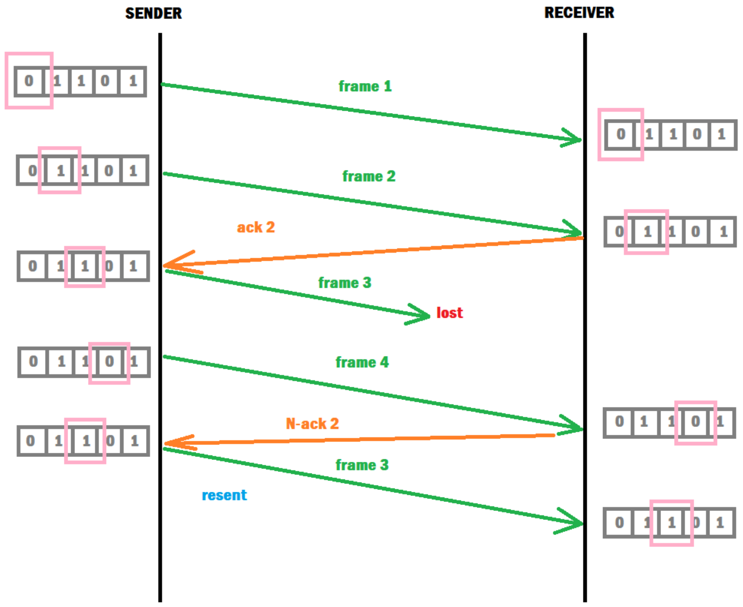
1. Selective Repeat.
2. Go Back N

Theory:

Selective Repeat and Go-Back-N are two common error control mechanisms used in the Data Link Layer of computer networks.:

1. Go-Back-N: In Go-Back-N error control mechanism, the sender sends multiple frames (packets) to the receiver before waiting for an acknowledgment. The receiver then checks each received frame for errors and sends an acknowledgment (ACK) or negative acknowledgment (NAK) message to the sender. If the sender receives an ACK, it knows that the frames were received successfully and can send the next set of frames. However, if the sender receives a NAK or no response, it retransmits all the frames starting from the frame that caused the error.
2. Selective Repeat: In Selective Repeat error control mechanism, the sender sends multiple frames to the receiver, and the receiver checks each frame for errors and sends an ACK or NAK message to the sender. If the receiver detects an error in a specific frame, it sends a NAK message to the sender requesting the retransmission of only that frame. The sender can then retransmit only the requested frame, while the remaining frames are left untouched. This mechanism can be more efficient than Go-Back-N, especially if only a small number of frames are lost or damaged.

Diagram:



Code:

<u>Selective Repeat:</u>

```
#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 window size: 3
Enter number of frames to transmit: 5
Enter 5 frames: 12 5 89 4 6
With sliding window protocol the frames will be sent in the following manner (assuming no
corruption of frames)

After sending 3 frames at each stage sender waits for acknowledgement sent by the receiver

12 5 89
Acknowledgement of above frames sent is received by sender

4 6
Acknowledgement of above frames sent is received by sender
```

Code:

Go Back N:

```
#include<stdio.h>
int main()
{
        int windowsize,sent=0,ack,i;
        printf("enter window size\n");
        scanf("%d",&windowsize);
        while(1)
        {
                for( i = 0; i < windowsize; i++)
                        {
                                printf("Frame %d has been transmitted.\n",sent);
                                sent++;
                if(sent == windowsize)
                    break;
            }
            printf("\nPlease enter the last Acknowledgement received.\n");
            scanf("%d",&ack);

            if(ack == windowsize)
                break;
            else
                sent = ack;
    }
return 0;
}
```

Output:

```
enter window size
8
Frame 0 has been transmitted.
Frame 1 has been transmitted.
Frame 2 has been transmitted.
Frame 3 has been transmitted.
Frame 4 has been transmitted.
Frame 5 has been transmitted.
Frame 6 has been transmitted.
Frame 7 has been transmitted.

Please enter the last Acknowledgement received.
2
Frame 2 has been transmitted.
Frame 3 has been transmitted.
Frame 4 has been transmitted.
Frame 5 has been transmitted.
Frame 6 has been transmitted.
Frame 7 has been transmitted.

Please enter the last Acknowledgement received.
8
```

Experiment 6:

Data Link Layer Error Detection Mechanism

1. Cyclic Redundancy Check

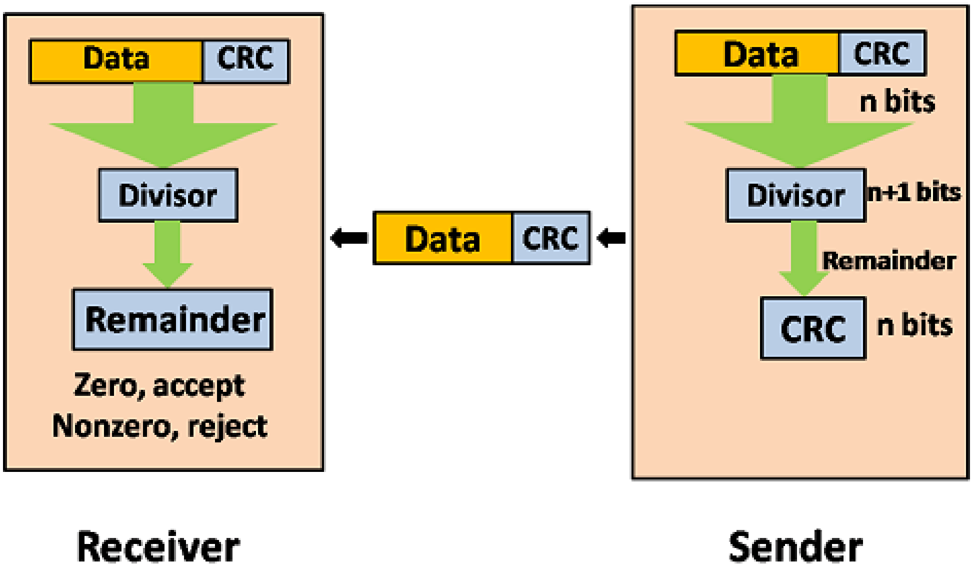
Theory:

Cyclic Redundancy Check (CRC) is a commonly used error detection mechanism in the Data Link Layer of computer networks. It involves appending a small sequence of redundant bits, called a CRC checksum, to the end of a block of data. The receiver then checks the received data and CRC checksum to detect any errors that may have occurred during transmission.

Here's how CRC works:

1. The sender generates a CRC checksum for the data block to be transmitted by performing a polynomial division of the data block by a predetermined generator polynomial.
2. The resulting remainder is appended to the end of the data block to form the transmitted frame.
3. The receiver performs the same polynomial division on the received data block (including the appended CRC checksum) using the same generator polynomial.
4. If the remainder obtained by the receiver is zero, then the received data is error-free. However, if the remainder is non-zero, then the receiver knows that the data block has been corrupted during transmission.
5. The receiver can then request retransmission of the corrupted data block.

Diagram:



Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
char* calculate(char* mes, const char* gen)
{
    int m = strlen(mes);
    int n = strlen(gen);
    char* message = (char*) malloc((m+n) * sizeof(char));
    strcpy(message, mes);
    strcat(message, "00000000");
    for (int i = 0; i <= m-n; i++)
    {
        if(message[i]!='0')
        {
            for (int j = 0; j < n; j++)
            {
                if(message[i+j] == gen[j])
                    message[i+j] = '0';
                else
                    message[i+j] = '1';
            }
        }
    }
    return message+m;
}
int main()
{
    char gen[] = "1011";
    char mes[100];
    printf("Enter message: ");
    scanf("%s", mes);
    char* crc = calculate(mes, gen);
    char* mesWithCRC = (char*) malloc((strlen(mes)+strlen(crc)+1) * sizeof(char));
    strcpy(mesWithCRC, mes);
    strcat(mesWithCRC, crc);
    char* rmessage = (char*) malloc((strlen(mes)+strlen(crc)+1) * sizeof(char));
    strcpy(rmessage, mesWithCRC);
    srand(time(0));
    int modulo = strlen(mesWithCRC);
    int errorIndex = rand() % modulo;

    // If the original bit was a 0, it is flipped to 1; otherwise, it is flipped to 0.
    if (mesWithCRC[errorIndex] == '0')
    {
        rmessage[errorIndex] = '1';
    }
    else
    {
        rmessage[errorIndex] = '0';
    }

    char* receivedMessage = (char*) malloc((strlen(mes)+1) * sizeof(char));
    strncpy(receivedMessage, rmessage, strlen(mes));
    receivedMessage[strlen(mes)] = '\0';
    char* receivedCRC = calculate(receivedMessage, gen);
    if (strcmp(receivedCRC, rmessage+strlen(mes)) == 0)
    {
        printf("No error detected!\n");
    }
    else
    {
        printf("Error detected.\n");
    }
    free(crc);
    free(mesWithCRC);
    free(rmessage);
    free(receivedMessage);
    return 0;
}
```

Output:

```
Enter message: 110101
No error detected!
free(): invalid pointer
Aborted
```