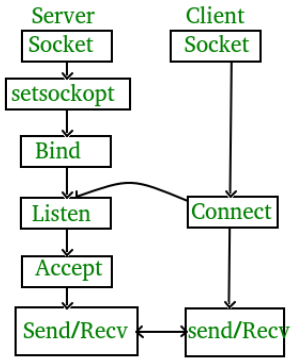
2)TCP socket programing

TCP (Transmission Control Protocol) is a connection-oriented protocol that guarantees reliable, ordered delivery of data between applications running on different hosts. In socket programming, TCP is one of the protocols that can be used to establish a reliable communication channel between a client and a server.



server.c

```
#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;
}
```

```
dinda@Sayantan:/mnt/c/Users/sayan/OneDrive/Desktop/New folder/Stop & Wait$ cd ..
dinda@Sayantan:/mnt/c/Users/sayan/OneDrive/Desktop/New folder$ cd TCP/
dinda@Sayantan:/mnt/c/Users/sayan/OneDrive/Desktop/New folder/TCP$ gcc 2server.c -o server
dinda@Sayantan:/mnt/c/Users/sayan/OneDrive/Desktop/New folder/TCP$ ./server
[+]TCP server socket created.
[+]Bind to the port number: 5566
Listening...
[+]Client connected.
Client: HELLO, THIS IS CLIENT.
Server: HI, THIS IS SERVER. HAVE A NICE DAY!!!
[+]Client disconnected.

[+]Client connected.
Client: HELLO, THIS IS CLIENT.
Server: HI, THIS IS SERVER. HAVE A NICE DAY!!!
[+]Client disconnected.
```

client.c

```
#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];
```

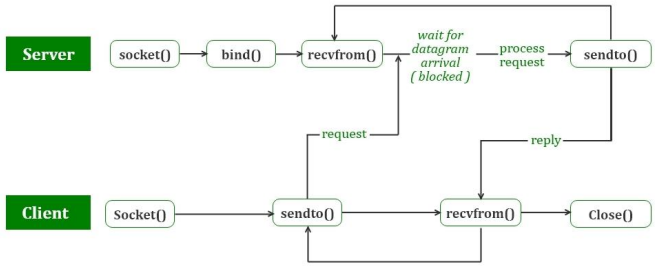
```
    int n;
    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0)
    {
        perror("[-]Socket error");
        exit(1);
    }
    printf("[+]TCP server socket created.\n");
    memset(&addr, '\0', sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = port;
    addr.sin_addr.s_addr = 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;
}
```

```
dinda@Sayantan:/mnt/c/Users/sayan/OneDrive/Desktop/New folder/TCP$ gcc 2client.c -o client1
dinda@Sayantan:/mnt/c/Users/sayan/OneDrive/Desktop/New folder/TCP$ ./client1
[+]TCP server socket created.
Connected to the server.
Client: HELLO, THIS IS CLIENT.
Server: HI, THIS IS SERVER. HAVE A NICE DAY!!!
Disconnected from the server.
dinda@Sayantan:/mnt/c/Users/sayan/OneDrive/Desktop/New folder/TCP$
```

```
dinda@Sayantan:/mnt/c/Users/sayan/OneDrive/Desktop/New folder/TCP$ gcc 2client.c -o client2
dinda@Sayantan:/mnt/c/Users/sayan/OneDrive/Desktop/New folder/TCP$ ./client2
[+]TCP server socket created.
Connected to the server.
Client: HELLO, THIS IS CLIENT.
Server: HI, THIS IS SERVER. HAVE A NICE DAY!!!
Disconnected from the server.
dinda@Sayantan:/mnt/c/Users/sayan/OneDrive/Desktop/New folder/TCP$
```

3)UDP socket programing

UDP (User Datagram Protocol) is a connectionless protocol that does not guarantee reliable, ordered delivery of data between applications running on different hosts. However, it is faster and more lightweight than TCP, making it a good choice for applications that require fast and efficient communication.



server.c

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

#define PORT 8888
#define BUF_SIZE 256

void error(char *msg)
{
   perror(msg);
   exit(1);
}

int main()
{
   printf("Server started\n");

   int sockfd;
   struct sockaddr_in servaddr, cliaddr;
   char buffer[BUF_SIZE];
   int n, len;

   sockfd = socket(AF_INET, SOCK_DGRAM, 0);
   if (sockfd < 0)
      error("ERROR opening socket");
```

```
    bzero(&servaddr, sizeof(servaddr));

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

    if (bind(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0)
        error("ERROR on binding");

    printf("Server waiting for broadcast messages...\n");

    while (1) {
        len = sizeof(cliaddr);
        n = recvfrom(sockfd, buffer, BUF_SIZE, 0, (struct sockaddr *)&cliaddr, &len);
        if (n < 0)
            error("ERROR in recvfrom");

        printf("Received broadcast message from %s:%d\n", inet_ntoa(cliaddr.sin_addr),
ntohs(cliaddr.sin_port));
        printf("Message: %s\n", buffer);
    }

    close(sockfd);

    return 0;
}
```

Output

```
dinda@Sayantan:/mnt/c/Users/sayan/OneDrive/Desktop/New folder/UDP$ ./server
Server started
Server waiting for broadcast messages...
Received broadcast message from 172.28.253.134:46278
Message: Broadcast message from client
Received broadcast message from 172.28.253.134:58169
Message: Broadcast message from client
```

client.c

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

```
#include <netinet/in.h>
#include <unistd.h>

#define BUF_SIZE 256
#define BROADCAST_PORT 8888

void error(char *msg)
{
   perror(msg);
   exit(1);
}

int main()
{
   printf("Client started\n");

   int sockfd;
   struct sockaddr_in broadcast_addr;
   char buffer[BUF_SIZE];
   int n, broadcastEnable = 1;

   sockfd = socket(AF_INET, SOCK_DGRAM, 0);
   if (sockfd < 0)
      error("ERROR opening socket");

   setsockopt(sockfd, SOL_SOCKET, SO_BROADCAST, &broadcastEnable, sizeof(broadcastEnable));

   bzero(&broadcast_addr, sizeof(broadcast_addr));
   broadcast_addr.sin_family = AF_INET;
   broadcast_addr.sin_port = htons(BROADCAST_PORT);
   broadcast_addr.sin_addr.s_addr = htonl(INADDR_BROADCAST);

   sprintf(buffer, "Broadcast message from client");

   n = sendto(sockfd, buffer, strlen(buffer), 0, (struct sockaddr *)&broadcast_addr,
sizeof(broadcast_addr));
   if (n < 0)
      error("ERROR in sendto");

   printf("Broadcast message sent\n");

   close(sockfd);

   return 0;
}
```

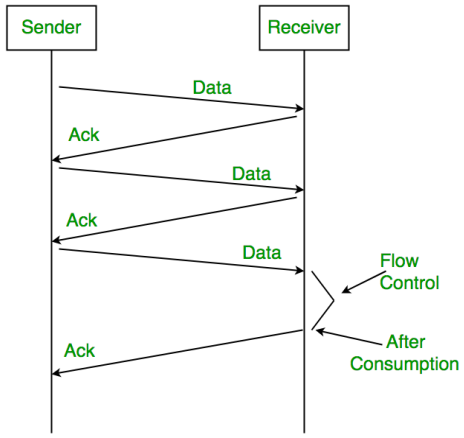
Output

```
dinda@Sayantan:/mnt/c/Users/sayan/OneDrive/Desktop/New folder/UDP$ ./client1
Client started
Broadcast message sent
dinda@Sayantan:/mnt/c/Users/sayan/OneDrive/Desktop/New folder/UDP$
```

```
dinda@Sayantan:/mnt/c/Users/sayan/OneDrive/Desktop/New folder/UDP$ ./client2
Client started
Broadcast message sent
```

4)a)Stop and Wait

Stop-and-wait protocol is a simple flow control protocol used in data communication systems. It is a technique where the sender sends a packet and waits for an acknowledgement (ACK) from the receiver before sending the next packet. The receiver, on the other hand, waits for a packet, acknowledges it, and then waits for the next packet.



server.c

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

#define PORT 8000
#define MAXLINE 1024

int main() {
    struct sockaddr_in servaddr, cliaddr;
    int sockfd, n;
    socklen_t len;
    char buffer[MAXLINE];
```

```
    char *ack = "ACK";
    int seq_no = 0;

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    memset(&servaddr, 0, sizeof(servaddr));

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

    bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr));

    printf("Server started...\n");

    while (1) {
        len = sizeof(cliaddr);
        n = recvfrom(sockfd, (char *)buffer, MAXLINE, MSG_WAITALL, (struct sockaddr *)&cliaddr, &len);
        buffer[n] = '\0';

        printf("Received message from client: %s\n", buffer);

        // check the sequence number
        if (buffer[0] == seq_no + '0') {
            printf("Packet %d received successfully\n", seq_no);
            sendto(sockfd, (const char *)ack, strlen(ack), MSG_CONFIRM, (const struct sockaddr *)&cliaddr,
len);
            printf("ACK %d sent to client\n", seq_no);
            seq_no = !seq_no; // toggle sequence number
        } else {
            printf("Packet %d lost\n", seq_no);
            printf("ACK %d not sent to client\n", seq_no);
        }
    }

    close(sockfd);
    return 0;
}
```

Output

```
dinda@Sayantan:/mnt/c/Users/sayan/OneDrive/Desktop/New folder/Stop & Wait$ gcc server.c -o server
dinda@Sayantan:/mnt/c/Users/sayan/OneDrive/Desktop/New folder/Stop & Wait$ ./server
Server started...
Received message from client: Packet 0
Packet 0 lost
ACK 0 not sent to client
```

client.c

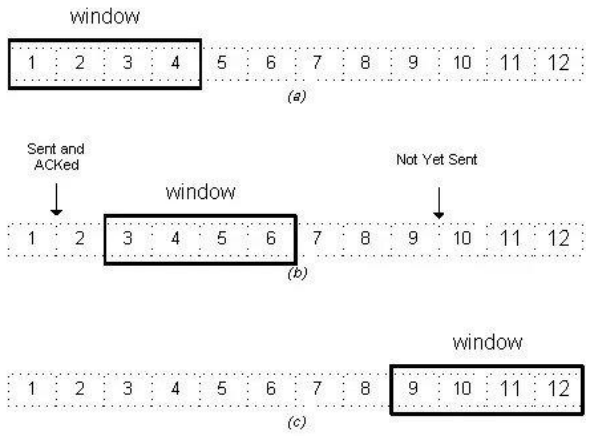
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>

#define PORT 8000
#define MAXLINE 1024

int main() {
    struct sockaddr_in servaddr;
    int sockfd, n;
    socklen_t len;
    char buffer[MAXLINE];
    char *msg1 = "Packet 0";
    char *msg2 = "Packet 1";
    char *ack = "ACK";

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    memset(&servaddr, 0, sizeof(servaddr));

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

    printf("Sending packet 0\n");
    sendto(sockfd, (const char *)msg1, strlen(msg1), MSG_CONFIRM, (const struct sockaddr *)&servaddr,
sizeof(servaddr));

    len = sizeof(servaddr);
    n = recvfrom(sockfd, (char *)buffer, MAXLINE, MSG_WAITALL, (struct sockaddr *)&servaddr, &len);
    buffer[n] = '\0';

    printf("Received ACK 0 from server\n");

    printf("Sending packet 1\n");
    sendto(sockfd, (const char *)msg2, strlen(msg2), MSG_CONFIRM, (const struct sockaddr *)&servaddr,
sizeof(servaddr));

    len = sizeof(servaddr);
    n = recvfrom(sockfd, (char *)buffer, MAXLINE, MSG_WAITALL, (struct sockaddr *)&servaddr, &len);
    buffer[n] = '\0';

    printf("Received ACK 1 from server\n");

    close(sockfd);
    return 0;
}
```

```
souptik@LAPTOP-HEBFCC8I:~$ cd /mnt/c/Users/Souptik\ Ghosh/Downloads
souptik@LAPTOP-HEBFCC8I:/mnt/c/Users/Souptik Ghosh/Downloads$ gcc 4client.c
souptik@LAPTOP-HEBFCC8I:/mnt/c/Users/Souptik Ghosh/Downloads$ ./a.out
Sending packet 0
```

4)b)Sliding Window

The sliding window protocol is a technique used in computer networking to efficiently and reliably transmit data over a network. It is a flow control protocol that manages the flow of packets between two network devices, typically a sender and a receiver.In a sliding window protocol, the sender and receiver agree on a window size, which determines the maximum number of packets that can be sent without receiving an acknowledgement. The sender then sends a block of packets to the receiver and waits for an acknowledgement. Once the acknowledgement is received, the sender moves the window forward and sends the next block of packets.



Code

```c
#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

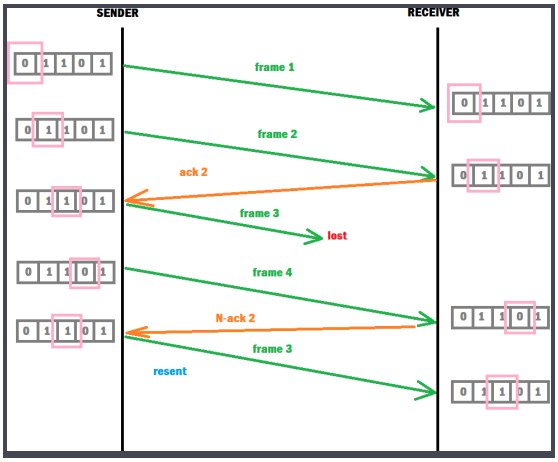
Acknowledgement of above frames sent is received by sender

4 6
Acknowledgement of above frames sent is received by sender

5)a)Selective Repeat

The Selective Repeat protocol is a flow control protocol used in computer networking to manage the flow of packets between two network devices, typically a sender and a receiver. It is an alternative to the Go-Back-N protocol.In Selective Repeat protocol, the sender maintains a window of unacknowledged packets, but unlike Go-Back-N, if a

packet is lost or damaged, only that packet is retransmitted, rather than all the packets following it.



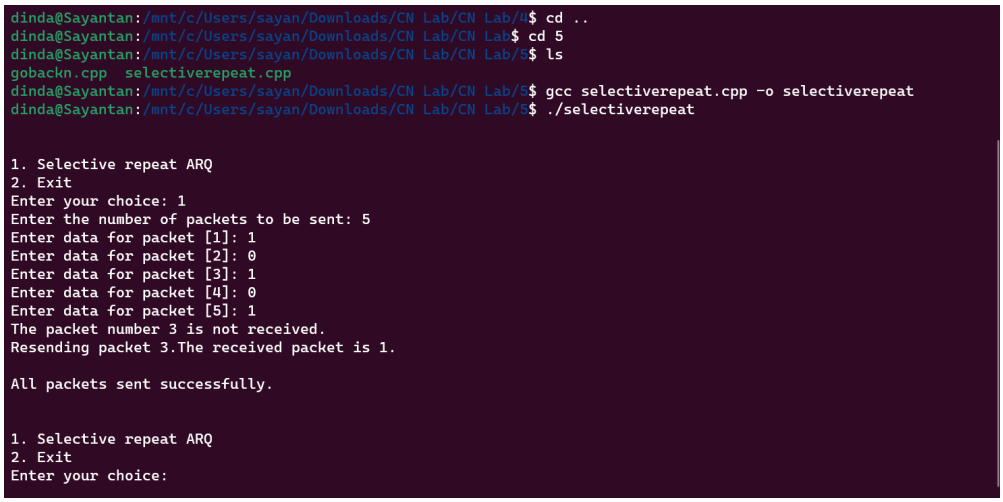
Code

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

int n, r;

struct frame {
   char ack;
   int data;
} frm[10];

int sender(void);
void recvack(void);
void resend(void);
//void resend1(void);
void goback(void);
void selective(void);

int main() {
   int c;

   do {
      printf("\n\n1. Selective repeat ARQ\n2. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &c);

      switch (c) {
         case 1:
            selective();
```

```
            break;

        case 2:
            exit(0);
            break;

        default:
            printf("Invalid choice! Try again.\n");
      }
   } while (c != 2);

   return 0;
}


void selective() {
   sender();
   recvack();
   resend();
   printf("\nAll packets sent successfully.\n");
}

int sender() {
   int i;

   printf("Enter the number of packets to be sent: ");
   scanf("%d", &n);

   for (i = 1; i <= n; i++) {
      printf("Enter data for packet [%d]: ", i);
      scanf("%d", &frm[i].data);
      frm[i].ack = 'y';
   }

   return 0;
}

void recvack() {
   int i;

   srand(time(NULL));
   r = rand() % n;

   frm[r].ack = 'n';

   for (i = 1; i <= n; i++) {
      if (frm[i].ack == 'n') {
```

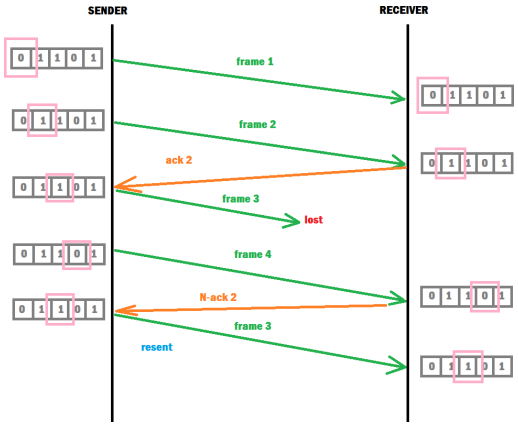
```
            printf("The packet number %d is not received.\n", r);
            break;
        }
    }
}

void resend() {  // SELECTIVE REPEAT
    printf("Resending packet %d.", r);
    sleep(2);
    frm[r].ack = 'y';
    printf("The received packet is %d.\n", frm[r].data);
}
```

Output



5)b)Go Back N

"Go-back-n" is a type of error control mechanism used in data communication protocols, particularly in the Transmission Control Protocol (TCP). It is a form of automatic repeat request (ARQ) in which the sender transmits a continuous stream of packets without waiting for an acknowledgement from the receiver for each packet.Overall, go-back-n is a simple and efficient error control mechanism that can handle errors and losses in network transmissions, but it can also result in unnecessary retransmissions in cases of minor packet losses.



Code:

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

int n, r;

struct frame {
   char ack;
   int data;
} frm[10];

int sender(void);
void recvack(void);
//void resend(void);
void resend1(void);
void goback(void);
void selective(void);

int main() {
   int c;

   do {
      printf("\n\n1. Go-Back-N ARQ\n2. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &c);

      switch (c) {
         case 1:
            goback();
            break;

         case 2:
            exit(0);
```

```
            break;

        default:
            printf("Invalid choice! Try again.\n");
      }
   } while (c != 2);

   return 0;
}

void goback() {
   sender();
   recvack();
   resend1();
   printf("\nAll packets sent successfully.\n");
}


int sender() {
   int i;

   printf("Enter the number of packets to be sent: ");
   scanf("%d", &n);

   for (i = 1; i <= n; i++) {
      printf("Enter data for packet [%d]: ", i);
      scanf("%d", &frm[i].data);
      frm[i].ack = 'y';
   }

   return 0;
}

void recvack() {
   int i;

   srand(time_t(NULL));
   r = rand() % n;

   frm[r].ack = 'n';

   for (i = 1; i <= n; i++) {
      if (frm[i].ack == 'n') {
         printf("The packet number %d is not received.\n", r);
         break;
      }
   }
```

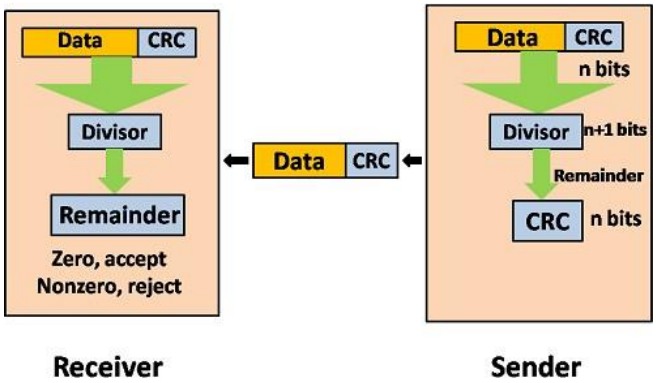
```
}


void resend1() { // GO-BACK-N
   int i;

   printf("Resending from packet %d.", r);

   for (i = r; i <= n; i++) {
      sleep(2);
      frm[i].ack = 'y';
      printf("Received data of packet %d is %d.\n", i, frm[i].data);
   }
}
```

Output

```
dinda@Sayantan:/mnt/c/Users/sayan/Downloads/CN Lab/CN Lab/5$ gcc gobackn.cpp -o gobackn
dinda@Sayantan:/mnt/c/Users/sayan/Downloads/CN Lab/CN Lab/5$ ./gobackn


1. Go-Back-N ARQ
2. Exit
Enter your choice: 1
Enter the number of packets to be sent: 3
Enter data for packet [1]: 1
Enter data for packet [2]: 0
Enter data for packet [3]: 1
The packet number 1 is not received.
Resending from packet 1.Received data of packet 1 is 1.
Received data of packet 2 is 0.
Received data of packet 3 is 1.

All packets sent successfully.


1. Go-Back-N ARQ
2. Exit
Enter your choice:
```

6)Cyclic Redundancy Check

Cyclic Redundancy Check (CRC) is a type of error detection technique used in digital communication networks, such as Ethernet and Wi-Fi, to detect errors in transmitted data. It works by adding a fixed-size check value, or CRC code, to a block of data before transmission. The receiver then calculates the CRC code for the received block of data and compares it with the CRC code that was transmitted. If the two codes match, the data is assumed to have been transmitted without errors.



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()
{
   printf("Kaninika Datta\n");
   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

```
dinda@Sayantan:/mnt/c/Users/sayan/Downloads/CN Lab/CN Lab$ gcc crc.c -o crc
dinda@Sayantan:/mnt/c/Users/sayan/Downloads/CN Lab/CN Lab$ ./crc
Enter message: abcd
Error detected.
free(): invalid pointer
Aborted
dinda@Sayantan:/mnt/c/Users/sayan/Downloads/CN Lab/CN Lab$
```