PROGRAM-5

SLIDING WINDOW FLOW CONTROL PROTOCOL

a)STOP AND WAIT PROTOCOL

Aim: To implement stop and wait flow control protocol

Stop and wait means, whatever the data that sender wants to send, he sends the data to the receiver. After sending the data, he stops and waits until he receives the acknowledgment from the receiver. The stop and wait protocol is a flow control protocol where flow control is one of the services of the data link layer.

It is a data-link layer protocol which is used for transmitting the data over the noiseless channels. It provides unidirectional data transmission which means that either sending or receiving of data will take place at a time. It provides flow-control mechanism but does not provide any error control mechanism.

The idea behind the usage of this frame is that when the sender sends the frame then he waits for the acknowledgment before sending the next frame.

If the sender doesn't receive ACK for previous sent packet after a certain period of time, the sender *times out* and *retransmits* that packet again. There are two cases when the sender doesn't receive ACK; One is when the ACK is lost and the other is when the frame itself is not transmitted.

Algorithm

Sender side

Rule 1: Sender sends one data packet at a time.

Rule 2: Sender sends the next packet only when it receives the acknowledgment of the previous packet.

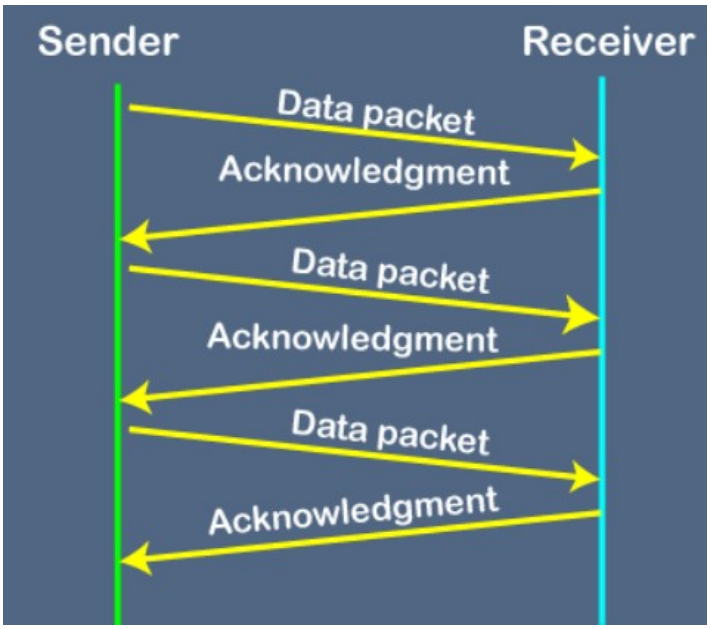
Therefore, the idea of stop and wait protocol in the sender's side is very simple, i.e., send one packet at a time, and do not send another packet before receiving the acknowledgment.

Receiver side

Rule 1: Receive and then consume the data packet.

Rule 2: When the data packet is consumed, receiver sends the acknowledgment to the sender.

Therefore, the idea of stop and wait protocol in the receiver's side is also very simple, i.e., consume the packet, and once the packet is consumed, the acknowledgment is sent. This is known as a flow control mechanism.



Client program

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
typedef struct packet{
    char data[1024];
}Packet;
typedef struct frame{
    int frame_kind; //ACK:0, SEQ:1 FIN:2
    int sq_no;
    int ack;
    Packet packet;
}Frame;
int main(int argc, char** argv){
```

```
	if (argc != 2){
		printf("Usage: %s <port>", argv[0]);
		exit(0);
	}
	int port = atoi(argv[1]);
	int sockfd;
	struct sockaddr_in serverAddr;
	char buffer[1024];
	socklen_t addr_size;
	int frame_id = 0;
	Frame frame_send;
	Frame frame_recv;
	int ack_recv = 1;
	sockfd = socket(AF_INET, SOCK_DGRAM, 0);
	memset(&serverAddr, '\0', sizeof(serverAddr));
	serverAddr.sin_family = AF_INET;
	serverAddr.sin_port = htons(port);
	serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
	while(1){
		if(ack_recv == 1){
			frame_send.sq_no = frame_id;
			frame_send.frame_kind = 1;
			frame_send.ack = 0;

			printf("Enter Data: ");
			scanf("%s", buffer);
			strcpy(frame_send.packet.data, buffer);
			sendto(sockfd, &frame_send, sizeof(Frame), 0, (struct
sockaddr*)&serverAddr, sizeof(serverAddr));
```

```
                printf("[%d]Frame Send\n",frame_id);
            }
            int addr_size = sizeof(serverAddr);
            int f_recv_size = recvfrom(sockfd, &frame_recv, sizeof(frame_recv), 0 ,(struct
sockaddr*)&serverAddr, &addr_size);

            if( f_recv_size > 0 && frame_recv.sq_no == 0 && frame_recv.ack == frame_id+1)
            {
                printf("[%d]Ack Received\n",frame_recv.ack);
                ack_recv = 1;
            }
          else
           {
                printf("[%d]Ack Not Received\n",frame_recv.ack);
                ack_recv = 0;
            }
            frame_id++;
        }

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

Output

Enter data: hi

[0] Frame Received

[0]Ack Received

Enter data: hello

[1]Frame Received

[0]Ack Recieved

Server_program

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <unistd.h>
#include <arpa/inet.h>
typedef struct packet{
   char data[1024];
}Packet;

typedef struct frame{
   int frame_kind; //ACK:0, SEQ:1 FIN:2
   int sq_no;
   int ack;
   Packet packet;
}Frame;

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

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

```
	int port = atoi(argv[1]);
	int sockfd;
	struct sockaddr_in serverAddr, newAddr;
	char buffer[1024];
	socklen_t addr_size;

	int frame_id=0;
	Frame frame_recv;
	Frame frame_send;

	sockfd = socket(AF_INET, SOCK_DGRAM, 0);

	memset(&serverAddr, '\0', sizeof(serverAddr));
	serverAddr.sin_family = AF_INET;
	serverAddr.sin_port = htons(port);
	serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");

	bind(sockfd, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
	addr_size = sizeof(newAddr);

	while(1){
		int f_recv_size = recvfrom(sockfd, &frame_recv, sizeof(Frame), 0, (struct sockaddr*)&newAddr, &addr_size);
		if (f_recv_size > 0 && frame_recv.frame_kind == 1 && frame_recv.sq_no == frame_id){
			printf("[%d]Frame Received: %s\n", frame_id,frame_recv.packet.data);

			frame_send.sq_no = 0;
```

```
                frame_send.frame_kind = 0;
                frame_send.ack = frame_recv.sq_no + 1;
                sendto(sockfd, &frame_send, sizeof(frame_send), 0, (struct
sockaddr*)&newAddr, addr_size);
                printf("[%d]Ack Send\n",frame_send.ack);
        }else{
                printf("[%d]Frame Not Received\n", frame_send.ack);
        }
        frame_id++;
    }

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

OUTPUT

```
[0]Frame Received: hi
[1]Ack Send
 [1]Frame Received: hello
[2]Ack Send
```

b) Go-Back--N ARQ

In Go-Back-N ARQ, **N** is the sender's window size. Suppose we say that Go-Back-3, which means that the three frames can be sent at a time before expecting the acknowledgment from the receiver. It uses the principle of protocol pipelining in which the multiple frames can be sent before receiving the acknowledgment of the first frame.

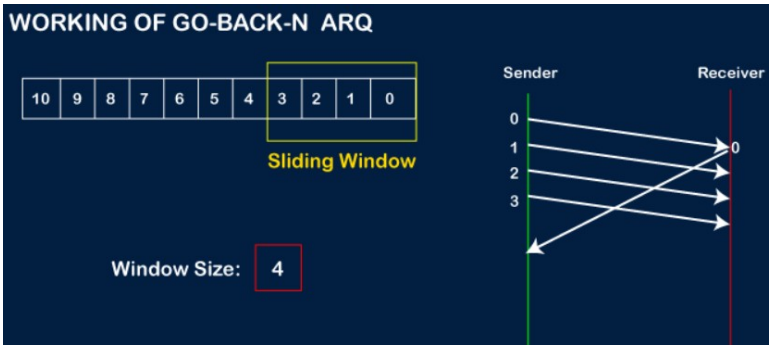
If we have five frames and the concept is Go-Back-3, which means that the three frames can be sent, i.e., frame no 1, frame no 2, frame no 3 can be sent before expecting the acknowledgment of frame no 1.

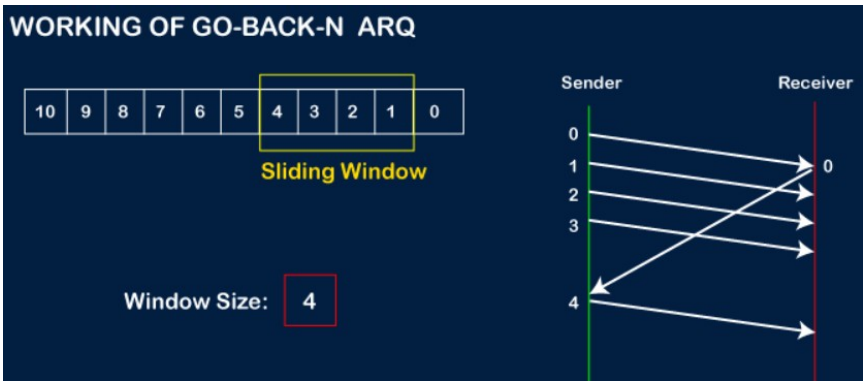
The number of frames that can be sent at a time totally depends on the size of the sender's window. So, we can say that 'N' is the number of frames that can be sent at a time before receiving the acknowledgment from the receiver.

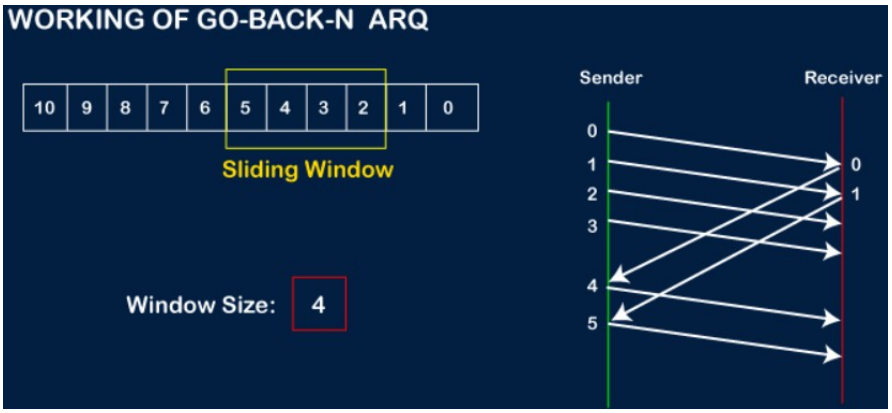
If the acknowledgment of a frame is not received within an agreed-upon time period, then all the frames available in the current window will be retransmitted. Suppose we have sent the frame no 5, but we didn't receive the acknowledgment of frame no 5, and the current window is holding three frames, then these three frames will be retransmitted.

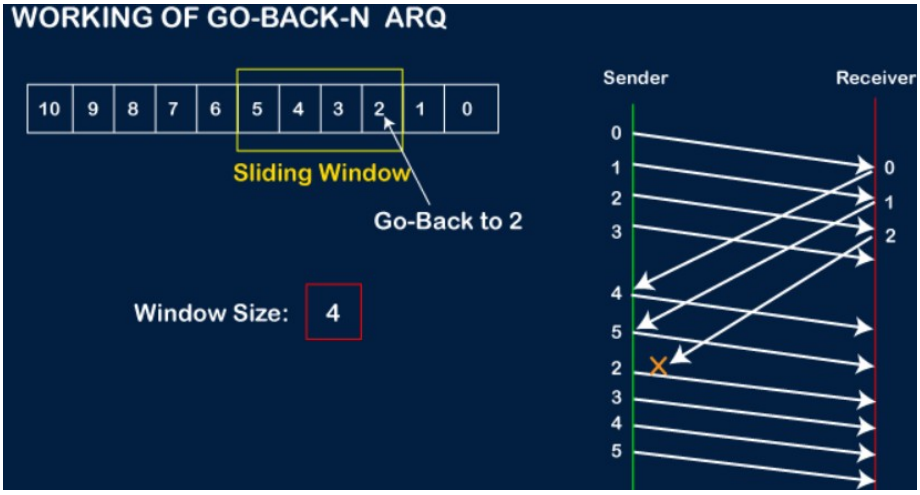
Important points related to Go-Back-N ARQ:

- In Go-Back-N, N determines the sender's window size, and the size of the receiver's window is always 1.
- It does not consider the corrupted frames and simply discards them.
- It does not accept the frames which are out of order and discards them.
- If the sender does not receive the acknowledgment, it leads to the retransmission of all the current window frames.









Algorithm

Sender:

Step1: Set the window size. Let it be 3.

Step2: Enter the number of frames. Let it be 5.

Step3: Send the first 3 frames equivalent to window size. Let it be 1 2 3

Step4: if acknowledgement for the first frame is received send the 4^{th} frame.

Step5: if acknowledgement for the second frame is not received, retransmit the window 2 3 4

Step5: The process is repeated until all frames are send and all are acknowledged.

Receiver:

Step1: If the frame is received, send acknowledgement for the frame.

Step2: If the frame is not received, send a negative acknowledgement and ask the sender to retransmit the window.

Step3: The process is repeated until all frames are received.

Client Program

```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include<stdlib.h>
typedef struct frame{
        int sq_no;
        int ack;
        }Frame;

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

          if (argc != 2){
                    printf("Usage: %s <port>", argv[0]);
                    exit(0);
          }
  int clientSocket, portNum, nBytes,nf;
  char buffer[1024];
  int port = atoi(argv[1]);
  int i=1;
  int n,j=1,k=0;
  struct sockaddr_in serverAddr;
  socklen_t addr_size;
  Frame fsend;
  /*Create UDP socket*/
  clientSocket = socket(AF_INET, SOCK_DGRAM, 0);
```

```
  /*Configure settings in address struct*/
  serverAddr.sin_family = AF_INET;
  serverAddr.sin_port = htons(port);
  serverAddr.sin_addr.s_addr = INADDR_ANY;
   /*Initialize size variable to be used later on*/
  addr_size = sizeof serverAddr;
   printf("enter the number of frames");
   scanf("%s",buffer);
    nf=buffer[0]-48;
    nBytes = strlen(buffer) + 1;
    sendto(clientSocket,buffer,1024,0,(struct sockaddr *)&serverAddr,addr_size);
    printf("enter the window size");
    scanf("%s",buffer);
sendto(clientSocket,buffer,1024,0,(struct sockaddr *)&serverAddr,addr_size);
    n=buffer[0]-48;
    while (i<=n)
        {
            fsend.sq_no=i;
            printf("\nFrame %d send",i);
            sendto(clientSocket,&fsend,sizeof(Frame),0,(struct sockaddr *)&serverAddr,addr_size);
            i++;
            j++;
        }
     while(k<nf)
     {
      recvfrom(clientSocket,&fsend,sizeof(fsend),0,(struct sockaddr *)&serverAddr, &addr_size);
       printf("\nwaiting for the acknowledgement");
         if(fsend.ack>0)
```

```
  {

   i=fsend.sq_no;
   printf("\nack %d recieved\n",fsend.ack);
   if(j<=nf)
     {
      printf("\nframe %d sent",j);
      fsend.sq_no=j;
      sendto(clientSocket,&fsend,sizeof(Frame),0,(struct sockaddr *)&serverAddr,addr_size);
      j++;
     }
   k++;
  }
else
  {
     printf("\nack %d not recieved",-fsend.ack);
    printf("\nRetransmitting window");
    fsend.sq_no=-fsend.ack;
    i=fsend.sq_no;
    do
      {
     printf("\nFrame %d send",i);
     fsend.sq_no=i;
     sendto(clientSocket,&fsend,sizeof(Frame),0,(struct sockaddr *)&serverAddr,addr_size);
     i++;
    }while(i<j);
  }
```

```
    }

    return 0;
```

```
nislinux@LAPTOP-QAVB4LA7:~/my$ ./a.out 6005
enter the number of frames5
enter the window size2

Frame 1 send
Frame 2 send
waiting for the acknowledgement
ack 1 not recieved
Retransmitting window
Frame 1 send
Frame 2 send
ack 1 recieved
frame 3 sent
ack 2 recieved
frame 4 sent
ack 3 recieved
frame 5 sent
ack 4 not recieved
Retransmitting window
Frame 4 send
Frame 5 send
ack 4 recieved
ack 5 not recieved
Retransmitting window
Frame 5 send
ack 5 not recieved
Retransmitting window
Frame 5 send
ack 5 recievednislinux@LAPTOP-QAVB4LA7:~/my$
```

```
}
```

Server Program

```
#include <stdio.h>

#include <sys/socket.h>

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

```
#include <string.h>
#include <stdlib.h>
#include<time.h>
#include<unistd.h>
typedef struct frame
{
   int sq_no;
   int ack;
 }Frame;
int main(int argc, char** argv){
        if (argc != 2){
                printf("Usage: %s <port>", argv[0]);
                exit(0);
        }
  int udpSocket, nBytes;
  int port = atoi(argv[1]);
  char buffer[1024];
  struct sockaddr_in serverAddr, clientAddr;
  socklen_t addr_size, client_addr_size;
  int i,nf,n,flag,k=0,m,p;
  srand(time(NULL));
  Frame frec;

  /*Create UDP socket*/
  if((udpSocket = socket(AF_INET, SOCK_DGRAM, 0))<0)
  {
   perror("error in socket creation");
```

```
 exit(0);
}


/*Configure settings in address struct*/
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(port);
serverAddr.sin_addr.s_addr = INADDR_ANY;

/*Bind socket with address struct*/
if((bind(udpSocket, (struct sockaddr *) &serverAddr, sizeof(serverAddr)))<0)
{
 perror("bind");
 exit(0);
}

/*Initialize size variable to be used later on*/
addr_size = sizeof(serverAddr);
recvfrom(udpSocket,buffer,1024,0,(struct sockaddr *)&serverAddr, &addr_size);
nf=buffer[0]-48;
recvfrom(udpSocket,buffer,1024,0,(struct sockaddr *)&serverAddr, &addr_size);
n=buffer[0]-48;m=n;
while(k<nf)
 {
 recvfrom(udpSocket,&frec,sizeof(Frame),0,(struct sockaddr *)&serverAddr, &addr_size);
 flag=rand()%2;
 if(!flag)
```

```
    {
        printf("\nFrame %d recieved\n",frec.sq_no);
        frec.ack=frec.sq_no;
        sendto(udpSocket,&frec,sizeof(frec),0,(struct sockaddr *)&serverAddr,addr_size);k+
+;m=m+1;
         if(m==nf+1)
          {
           m--;
          }
       }
   else
    {
     printf("\nFrame %d not recieved",frec.sq_no);
     printf("\n send negative acknowledgement\n retransmit the window");
     frec.ack=-frec.sq_no;
     i=-frec.ack;
     sendto(udpSocket,&frec,sizeof(frec),0,(struct sockaddr *)&serverAddr,addr_size);
     while(i<m)
      {
       recvfrom(udpSocket,&frec,sizeof(Frame),0,(struct sockaddr *)&serverAddr, &addr_size);
       i++;
      }
    }
 }
 return 0;
}
```

```
nislinux@LAPTOP-QAVB4LA7:~/my$ ./a.out 6005
{
Frame 1 not recieved
 send negative acknowledgement
 retransmit the window
Frame 1 recieved

Frame 2 recieved

Frame 3 recieved

Frame 4 not recieved
 send negative acknowledgement
 retransmit the window
Frame 4 recieved

Frame 5 not recieved
 send negative acknowledgement
 retransmit the window
Frame 5 not recieved
 send negative acknowledgement
 retransmit the window
Frame 5 recieved
}
```

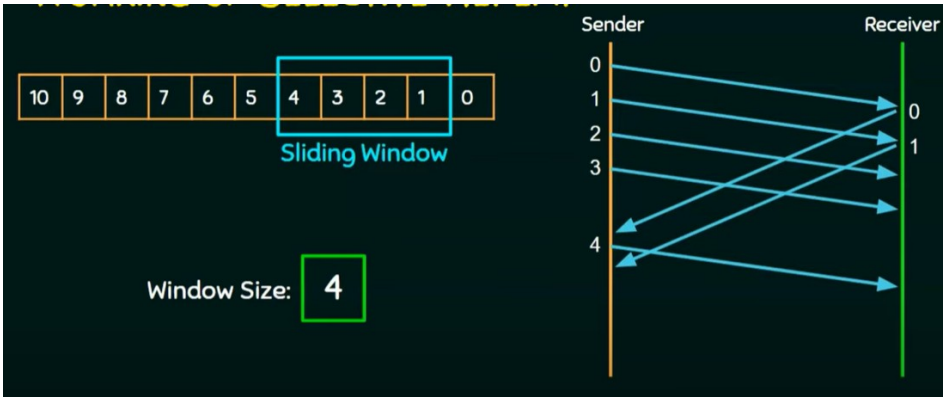
C) SELECTIVE REPEAT ARQ

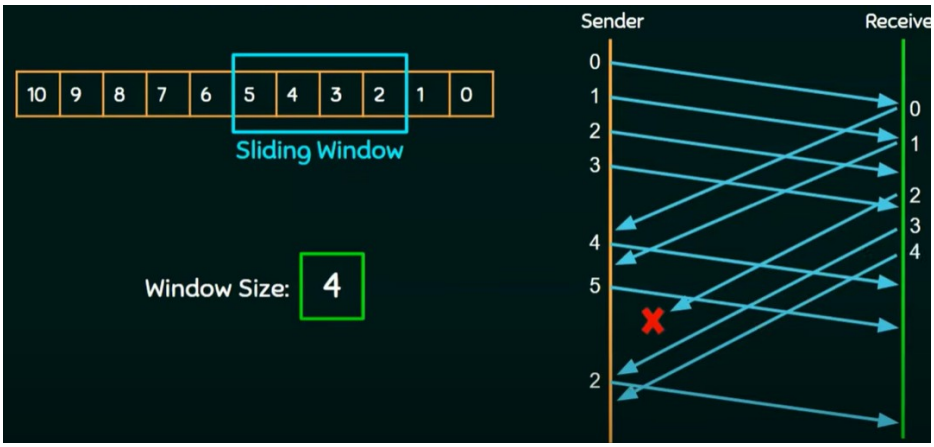
It is also known as Sliding Window Protocol and used for error detection and control in the data link layer.

In the selective repeat, the sender sends several frames specified by a window size even without the need to wait for individual acknowledgement from the receiver as in Go-Back-N ARQ. In selective repeat protocol, the retransmitted frame is received out of sequence.

In Selective Repeat ARQ only the lost or error frames are retransmitted, whereas correct frames are received and buffered.

The receiver while keeping track of sequence numbers buffers the frames in memory and sends NACK for only frames which are missing or damaged. The sender will send/retransmit a packet for which NACK is received.





Algorithm

Sender:

Step1: Set the window size. Let it be 3.

Step2: Enter the number of frames. Let it be 5.

Step3: Send the first 3 frames equivalent to window size. Let it be 1 2 3

Step4: if acknowledgement for the first frame is received send the 4th frame.

Step5: if acknowledgement for the second frame is not received, retransmit the frame 2 only.

Step5: The process is repeated until all frames are send and all are acknowledged.

Receiver:

Step1: If the frame is received, send acknowledgement for the frame.

Step2: If the frame is not received, send a negative acknowledgement and ask the sender to retransmit the corresponding frame.

Step3: The process is repeated until all frames are received

Sender

```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include<stdlib.h>
typedef struct frame{
        int sq_no;
        int ack;
        }Frame;

int main(int argc, char** argv){
            if (argc != 2){
                        printf("Usage: %s <port>", argv[0]);
                        exit(0);
            }
  int clientSocket, portNum, nBytes,nf;
  char buffer[1024];
  int port = atoi(argv[1]);
  int i=1;
  int n,j=1,k=0,m=0;
```

```
struct sockaddr_in serverAddr;
socklen_t addr_size;
Frame fsend;

/*Create UDP socket*/
clientSocket = socket(AF_INET, SOCK_DGRAM, 0);

/*Configure settings in address struct*/
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(port);
serverAddr.sin_addr.s_addr = INADDR_ANY;

/*Initialize size variable to be used later on*/
addr_size = sizeof serverAddr;

 printf("enter the number of frames");
 scanf("%s",buffer);
  nf=buffer[0]-48;
  nBytes = strlen(buffer) + 1;
  sendto(clientSocket,buffer,1024,0,(struct sockaddr *)&serverAddr,addr_size);
  printf("enter the window size");
  scanf("%s",buffer);
  n=buffer[0]-48;
  while (i<=n)
     {
        fsend.sq_no=i;
        printf("\nFrame %d send",i);
        sendto(clientSocket,&fsend,sizeof(Frame),0,(struct sockaddr *)&serverAddr,addr_size);
```

```
          i++;
          j++;
       }
 i=1;
fsend.sq_no=i;
   while(k<nf)
      {
       recvfrom(clientSocket,&fsend,sizeof(fsend),0,(struct sockaddr *)&serverAddr, &addr_size);
       printf("\nwaiting for the acknowledgement");
         if(fsend.ack>0)
          {

           i=fsend.sq_no;
           printf("\nack %d recieved\n",fsend.ack);
           if(j<=nf)
             {
              printf("\nframe %d sent",j);
              fsend.sq_no=j;
              sendto(clientSocket,&fsend,sizeof(Frame),0,(struct sockaddr *)&serverAddr,addr_size);
              j++;
             }
           i++;k++;
          }
         else
           {
            printf("\nack %d not recieved",-fsend.ack);
            printf("\nRetransmitting window");
```

```
            fsend.sq_no=-fsend.ack;
            i=fsend.sq_no;
             printf("\nFrame %d send",i);
             fsend.sq_no=i;
             sendto(clientSocket,&fsend,sizeof(Frame),0,(struct sockaddr *)&serverAddr,addr_size);
            }
          }
    return 0;
}
```

Output

```
enter the number of frames5
enter the window size3

Frame 1 send
Frame 2 send
Frame 3 send
waiting for the acknowledgement
ack 1 not recieved
Retransmitting window
Frame 1 send
waiting for the acknowledgement
ack 2 recieved

frame 4 sent
waiting for the acknowledgement
ack 3 recieved

frame 5 sent
waiting for the acknowledgement
ack 1 recieved

waiting for the acknowledgement
ack 4 not recieved
Retransmitting window
Frame 4 send
waiting for the acknowledgement
ack 5 not recieved
Retransmitting window
Frame 5 send
waiting for the acknowledgement
ack 4 recieved

waiting for the acknowledgement
ack 5 recieved
```

Reciever:

```
#include <stdio.h>

#include <sys/socket.h>

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

```
#include <string.h>
#include <stdlib.h>
#include<time.h>
#include<unistd.h>

typedef struct frame{

   int sq_no;
   int ack;
}Frame;
int main(int argc, char** argv){
        if (argc != 2){
                printf("Usage: %s <port>", argv[0]);
                exit(0);
        }
  int udpSocket, nBytes;
int port = atoi(argv[1]);
  char buffer[1024];
  struct sockaddr_in serverAddr, clientAddr;
  socklen_t addr_size, client_addr_size;
  int i,nf,flag,k=0;
  srand(time(NULL));
  Frame frec;

  /*Create UDP socket*/
  if((udpSocket = socket(AF_INET, SOCK_DGRAM, 0))<0)
   {
   perror("error in socket creation");
```

```
  exit(0);
 }
 /*Configure settings in address struct*/
 serverAddr.sin_family = AF_INET;
 serverAddr.sin_port = htons(port);
 serverAddr.sin_addr.s_addr = INADDR_ANY;

 /*Bind socket with address struct*/
 if((bind(udpSocket, (struct sockaddr *) &serverAddr, sizeof(serverAddr)))<0)
  {
   perror("bind");
   exit(0);
  }

 /*Initialize size variable to be used later on*/
 addr_size = sizeof(serverAddr);
 recvfrom(udpSocket,buffer,1024,0,(struct sockaddr *)&serverAddr, &addr_size);
 nf=buffer[0]-48;
while(k<nf)
   {
   recvfrom(udpSocket,&frec,sizeof(Frame),0,(struct sockaddr *)&serverAddr, &addr_size);
  flag=rand()%2;
  if(!flag)
     {
      printf("\nFrame %d recieved\n",frec.sq_no);
      frec.ack=frec.sq_no;
      sendto(udpSocket,&frec,sizeof(frec),0,(struct sockaddr *)&serverAddr,addr_size);k++;
     }
```

```
    else
      {
       printf("\nFrame %d not recieved",frec.sq_no);
       printf("\n send negative acknowledgement\n retransmit the window");
       frec.ack=-frec.sq_no;
       sendto(udpSocket,&frec,sizeof(frec),0,(struct sockaddr *)&serverAddr,addr_size);
      }
  }
  return 0;
}
```

Output

```
Frame 1 not recieved
 send negative acknowledgement
 retransmit the window
Frame 2 recieved

Frame 3 recieved

Frame 1 recieved

Frame 4 not recieved
 send negative acknowledgement
 retransmit the window
Frame 5 not recieved
 send negative acknowledgement
 retransmit the window
Frame 4 recieved

Frame 5 recieved
```