Computer Networks

Project Report

“Reliable File Transfer Over UDP”

<u>Group Members</u>

Name	**CMS IDs**
Saira Fatima	227414
Muhammad Ubaid Ullah	211321

Contents

Abstract:

The idea is to transfer a video file “Reliably” over a UDP connection. Normally UDP is “Best Effort” protocol. This means that it does not guarantee any reliability on the transport layer. This makes it faster than TCP but at the same time not reliable to send important data. So, the goal is to make the UDP reliable manually by adding the characteristics of a TCP protocol.

Advantages of using UDP:

As there’s less computation regarding any reliability, therefore, UDP serves as a faster protocol to send or receive the data. Both the sender and receiver ends have less computation complexity as compared to a TCP connection; therefore, it is faster in speed.

Disadvantages of using UDP:

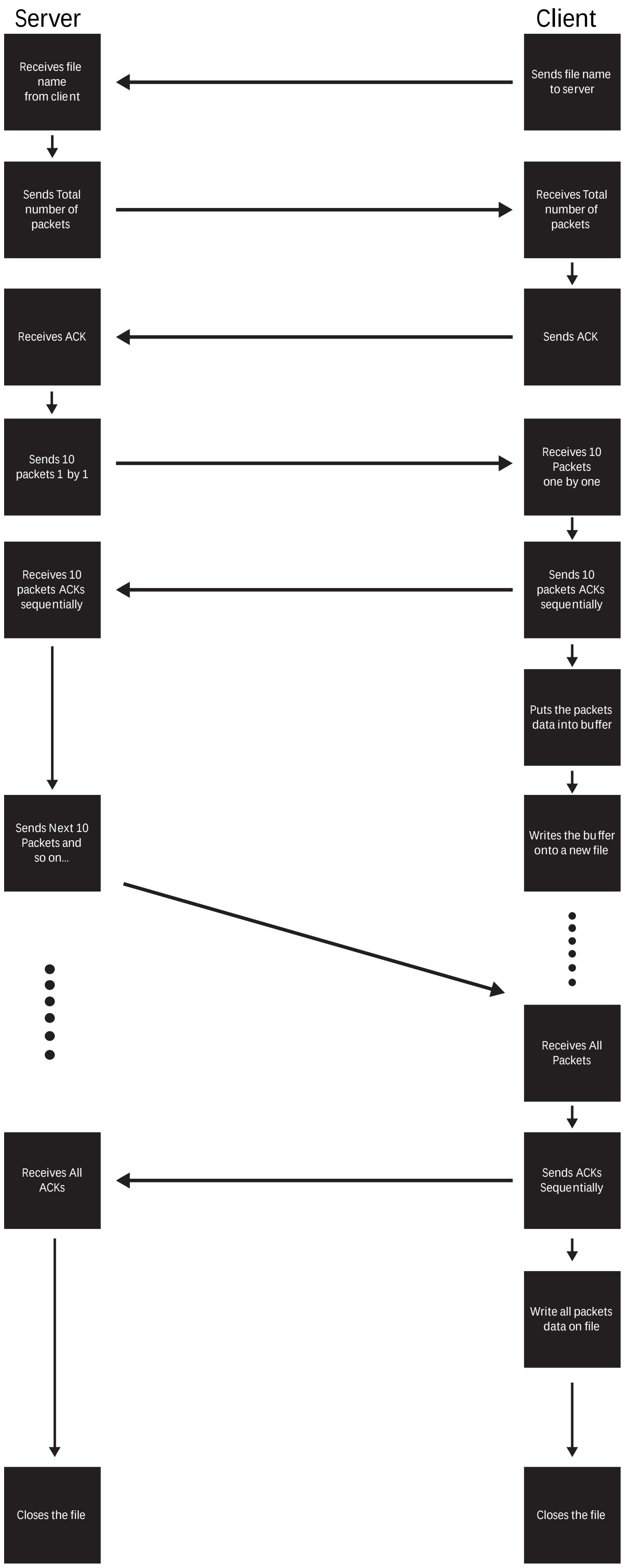
The main disadvantage is that it does not provide any guarantee of reliable transfer of data. Therefore, it is known as the best effort protocol. So, it is to be used where a compromise can be made on data loss.

Key Features of Project:

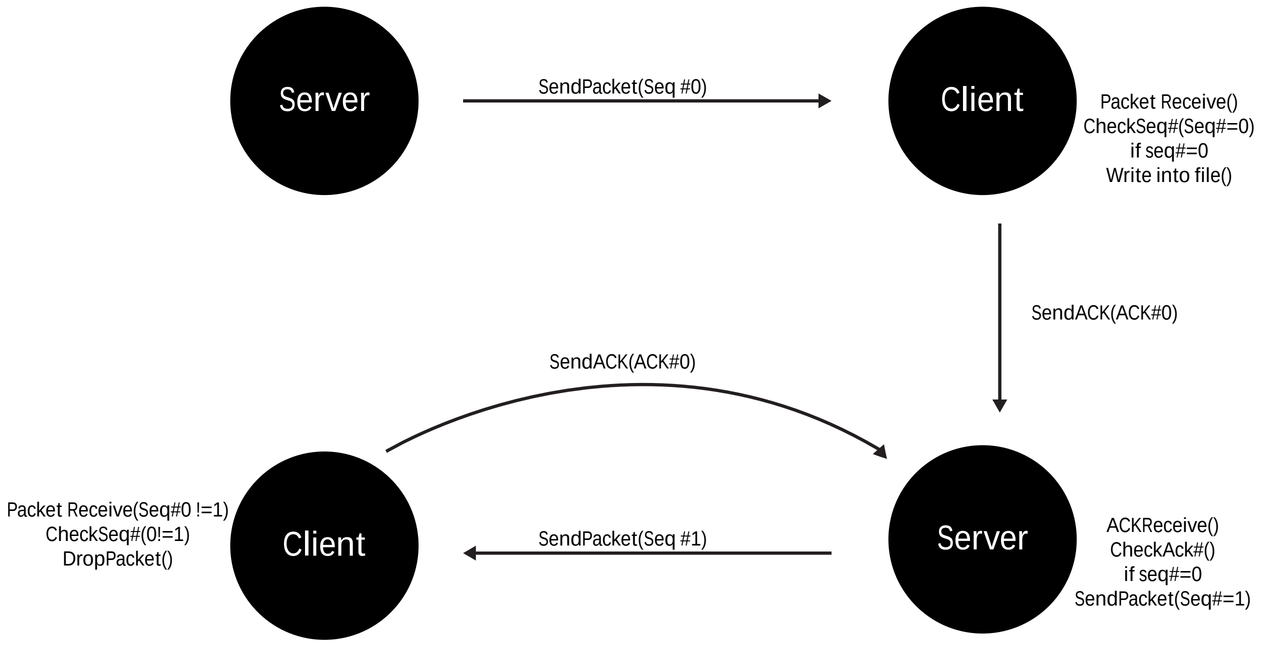
Following are the key features of our project;

1. UDP Protocol
2. Sequence Numbers
3. Packet Acknowledgements
4. Retransmission of dropped packet (Selective Repeat)
5. File saving using new name
6. Error handling
7. 512-byte data per packet
8. 10 packets transmission at a time

Process Diagram (Normal Flow):

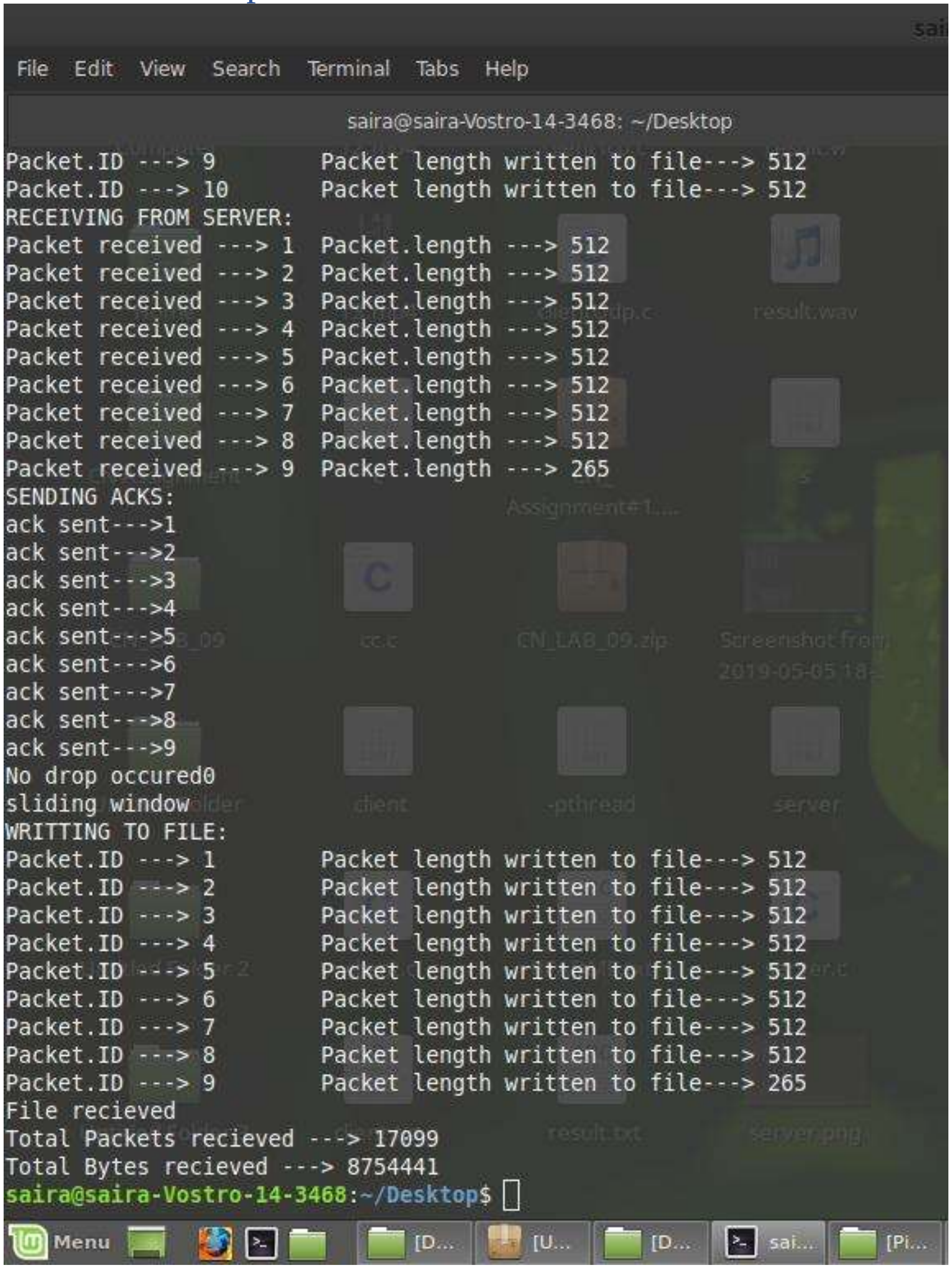

Server
Client
Receives file name from client
Sends file name to server
Sends Total number of packets
Receives Total number of packets
Receives ACK
Sends ACK
Sends 10 packets 1 by 1
Receives 10 Packets one by one
Receives 10 packets ACKs sequentially
Sends 10 packets ACKs sequentially
Puts the packets data into buffer
Sends Next 10 Packets and so on...
Writes the buffer onto a new file
Receives All Packets
Receives All ACKs
Sends ACKs Sequentially
Write all packets data on file
Closes the file
Closes the file

Finite State Machine FSM:



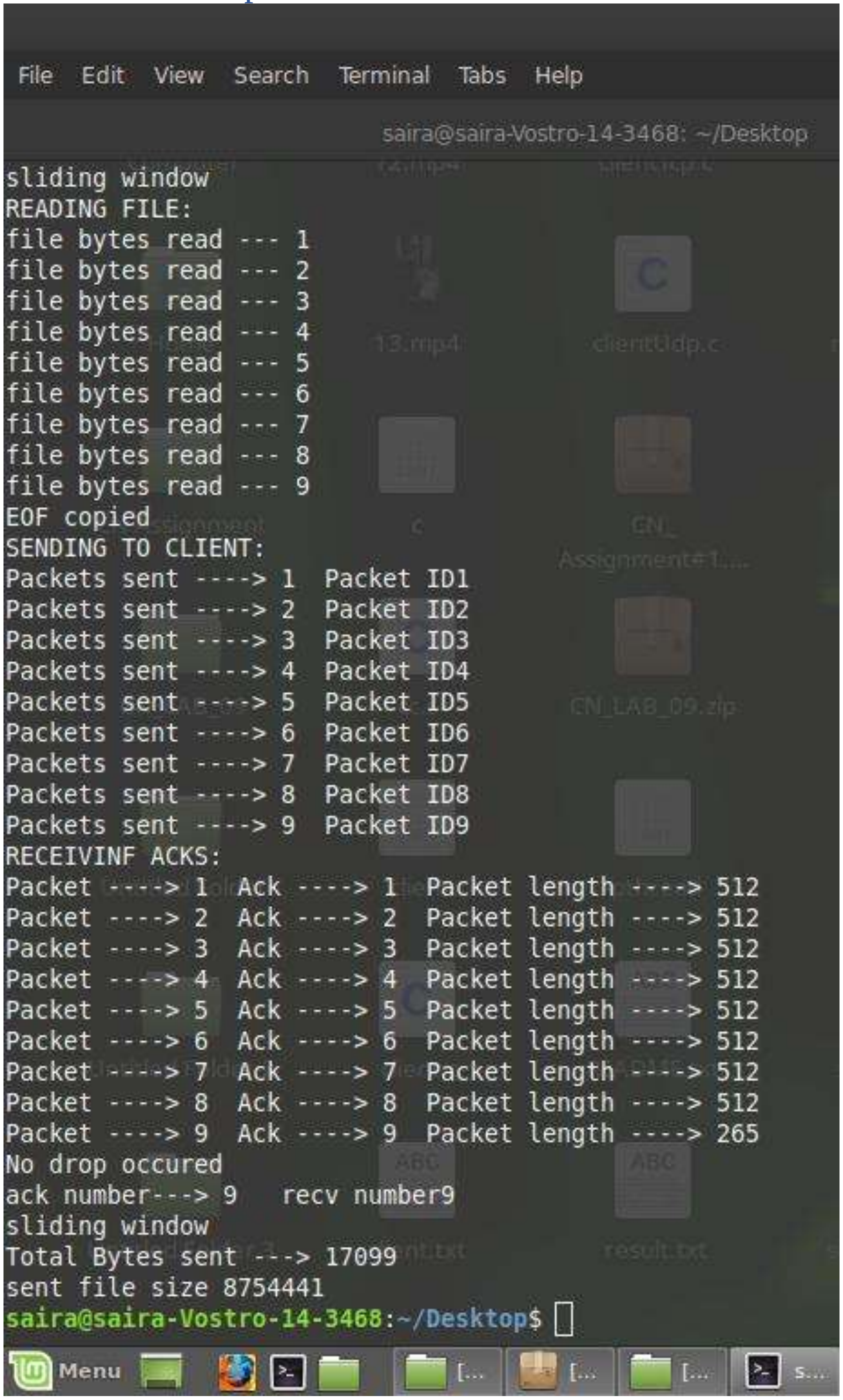
Note: For simplicity only seq # 0 and 1 is used here.

Client-Side Output:


File Edit View Search Terminal Tabs Help
saira@saira-Vostro-14-3468: ~/Desktop

```
Packet.ID ---> 9        Packet length written to file---> 512
Packet.ID ---> 10       Packet length written to file---> 512
RECEIVING FROM SERVER:
Packet received ---> 1  Packet.length ---> 512
Packet received ---> 2  Packet.length ---> 512
Packet received ---> 3  Packet.length ---> 512
Packet received ---> 4  Packet.length ---> 512
Packet received ---> 5  Packet.length ---> 512
Packet received ---> 6  Packet.length ---> 512
Packet received ---> 7  Packet.length ---> 512
Packet received ---> 8  Packet.length ---> 512
Packet received ---> 9  Packet.length ---> 265
SENDING ACKS:
ack sent--->1
ack sent--->2
ack sent--->3
ack sent--->4
ack sent--->5
ack sent--->6
ack sent--->7
ack sent--->8
ack sent--->9
No drop occured0
sliding window
WRITTING TO FILE:
Packet.ID ---> 1        Packet length written to file---> 512
Packet.ID ---> 2        Packet length written to file---> 512
Packet.ID ---> 3        Packet length written to file---> 512
Packet.ID ---> 4        Packet length written to file---> 512
Packet.ID ---> 5        Packet length written to file---> 512
Packet.ID ---> 6        Packet length written to file---> 512
Packet.ID ---> 7        Packet length written to file---> 512
Packet.ID ---> 8        Packet length written to file---> 512
Packet.ID ---> 9        Packet length written to file---> 265
File recieved
Total Packets recieved ---> 17099
Total Bytes recieved ---> 8754441
saira@saira-Vostro-14-3468:~/Desktop$
```

Server-Side Output:


File Edit View Search Terminal Tabs Help
saira@saira-Vostro-14-3468: ~/Desktop

```
sliding window
READING FILE:
file bytes read --- 1
file bytes read --- 2
file bytes read --- 3
file bytes read --- 4
file bytes read --- 5
file bytes read --- 6
file bytes read --- 7
file bytes read --- 8
file bytes read --- 9
EOF copied
SENDING TO CLIENT:
Packets sent ----> 1  Packet ID1
Packets sent ----> 2  Packet ID2
Packets sent ----> 3  Packet ID3
Packets sent ----> 4  Packet ID4
Packets sent ----> 5  Packet ID5
Packets sent ----> 6  Packet ID6
Packets sent ----> 7  Packet ID7
Packets sent ----> 8  Packet ID8
Packets sent ----> 9  Packet ID9
RECEIVINF ACKS:
Packet ----> 1  Ack ----> 1  Packet length ----> 512
Packet ----> 2  Ack ----> 2  Packet length ----> 512
Packet ----> 3  Ack ----> 3  Packet length ----> 512
Packet ----> 4  Ack ----> 4  Packet length ----> 512
Packet ----> 5  Ack ----> 5  Packet length ----> 512
Packet ----> 6  Ack ----> 6  Packet length ----> 512
Packet ----> 7  Ack ----> 7  Packet length ----> 512
Packet ----> 8  Ack ----> 8  Packet length ----> 512
Packet ----> 9  Ack ----> 9  Packet length ----> 265
No drop occured
ack number---> 9   recv number9
sliding window
Total Bytes sent ---> 17099
sent file size 8754441
saira@saira-Vostro-14-3468:~/Desktop$
```

Client-Side Code:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include <dirent.h>
#include <stdbool.h>
#define PORT    8002
#define BUF_SIZE 512 //Max buffer size of the data in a frame

/*A frame packet with unique id, length and data and acks*/
struct pkt {
            long int ID;
            long int length;
            char data[BUF_SIZE];
   long int ack;
};


int main(){
            struct sockaddr_in servddr, cliaddr;      //server
            struct stat st;
            struct pkt pkt_test; //pkt for testing
            struct timeval t_out = {0, 0};
   struct pkt pkta[11];  //array of 10 pkts of type "pkt"
            char   cmd_send[50]; // buffer to send file name
   ssize_t length;//length of cl_addr
   long int total_Packets = 0; //variable to store number of packets to be sent
   long int bytes_rec = 0; //variable to store total Bytes received when file is received completely
            int    sockfd = 0; //server descriptors
   int    pktrecv=0; //variable to store number of paktets received from server
            FILE   *fptr; //file pointer
   int    count=1;//testing variable
   bool   check[11]={false};
   int    drop_flag=0;
  int rcv_num=1;
  int ack_num=1;
   int    resnd=0;
  int resend_frame=0;
int         t_out_flag = 0;
```

```
/****************************************************************************************
*****/

          /*Clear all the data buffer and structure*/
          memset(&servddr, 0, sizeof(servddr));
          memset(& cliaddr, 0, sizeof( cliaddr));

         /**************************************************/
          /*Populate servddr structure with IP address and Port*/
          servddr.sin_family = AF_INET;
          servddr.sin_port =htons(PORT);
          servddr.sin_addr.s_addr = INADDR_ANY;

          /**************************************************/


          if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
                    perror("Socket creation failed");
   else{
      printf("Socket Created:\n");
      }

/****************************************************************************************
***/
      memset(cmd_send, 0, sizeof(cmd_send)); //cmd_send is buffer used to send file name

                    printf("Enter your file name:\n");
                    scanf(" %[^\n]%*c", cmd_send);//taking file name as input


      //sending file name
             if (sendto(sockfd, cmd_send, sizeof(cmd_send), 0, (struct sockaddr *) &servddr,
sizeof(servddr)) == -1) {
                              perror("File name sending failed");
      }
           /**************************************************/



             t_out.tv_sec = 2;
                    setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, (char *)&t_out, sizeof(struct
timeval));          //Enable the timeout option if server does not respond
            /**************************************************/

             //Receive number of packets to be sent
      recvfrom(sockfd, &(total_Packets), sizeof(total_Packets), 0, (struct sockaddr *) & cliaddr, (socklen_t
*) &length); //Get the total number of frame to recieve
             t_out.tv_sec = 0;
      setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, (char *)&t_out, sizeof(struct timeval));
          //Disable the timeout option

              if (total_Packets > 0) {
```

```
            //ack of paktes number which are received earlier
                        sendto(sockfd, &(total_Packets), sizeof(total_Packets), 0, (struct sockaddr *)
&servddr, sizeof(servddr));
                        printf("Total Packtets to be sent----> %ld\n", total_Packets);

                              fptr = fopen("result.wav", "wb");          //open the file in write mode

                              /*Recieve all the frames with window size 10 and send the acknowledgement
according to pkt ID*/

            pkt_test.length=512;
            /*when while loop starts
            /pktrecv is zero,total Packets are number of packets to  be sent
             */

                              while(total_Packets!=pktrecv)
                     {
                 ack_num=0;
                                                memset(&pkt_test, 0, sizeof(pkt_test));
                 //window size is 10 so we will runt this loop ,reveive 10 packets and then wait
                 printf("RECEIVING FROM SERVER:\n");
                 for(int k=1;k<=10;k++)
                 {

                                                  if((recvfrom(sockfd, &(pkta[k]), sizeof(pkta[k]), 0, (struct
sockaddr *) & cliaddr, (socklen_t *) &length))>0);  //Recieve the frame
                   {
                    printf("Packet received ---> %ld Packet.length ---> %ld\n", pkta[k].ID,pkta[k].length);
                    pktrecv+=1;  //increment it with every pkt received
                    if( check[pkta[k].ID]==true){
                    check[pkta[k].ID]=false;}
                   }


                   if( pkta[k].length!=512){ //if last packets is received befor k=10
                    break;
                    }
                   }







                 //10 packets are received
             /*************************************************/
                 //now send ID of received pkts as ack
                 printf("SENDING ACKS:\n");
                 for(int k=1;k<=10;k++){
```

```
                                                    if((sendto(sockfd, &(pkta[k].ID), sizeof(pkta[k].ID), 0, (struct
sockaddr *) &servddr, sizeof(servddr)))>0){       //Send the ack
                         printf("ack sent--->%ld\n", pkta[k].ID);
                          if(check[pkta[k].ID]=true){
                          check[pkta[k].ID]=true;}
                         ack_num+=1;
                     }
                     //if file is completely received before k=10 we have to break the loop
                    if( pkta[k].length!=512){ //if last packets is received befor k=10
                      break;
                    }

                  }
                                  for (int i=1;i<11;i++){

                   if(check[i]==false){
                    printf("flase flag\n");

                    resnd=i;
                      ack_num=resnd;
                    }
                  }



                  if(resnd==0){
                  printf("No drop occured%ld\n",drop_flag);

                   }
                  if(resnd!=0){
                  printf("PKT LOSS%ld\n",resnd);
                   exit(1);
                  }

                  for (int i=1;i<11;i++){
                      check[i]==false;}
                  //acks are sent accordinf to pkt ID
             /***********************************************/


                 recvfrom(sockfd, &( rcv_num), sizeof(rcv_num), 0, (struct sockaddr *) & cliaddr, (socklen_t
*) &length);
                 sendto(sockfd, &(ack_num), sizeof(ack_num), 0, (struct sockaddr *) &servddr,
sizeof(servddr));

                while (ack_num != rcv_num)  //Check for ack
                                                  {
                                                            //keep retrying until the ack matches

                                                            recvfrom(sockfd, &(pkta[ack_num].data),
sizeof(pkta[ack_num].data), 0, (struct sockaddr *) &cliaddr, (socklen_t *) &length);
```

```
                        sendto(sockfd, &(rcv_num), sizeof(rcv_num), 0, (struct sockaddr *) &servddr,
sizeof(servddr));
                                                        printf("ack ---> %ld       dropped, %d times\n");

                                                        resend_frame++;

                                                        printf("frame ---> %ld   dropped, %d times\n");

                                                        //Enable the timeout flag even if it fails after 200 tries
                                                        if (resend_frame == 200) {
                                                                t_out_flag = 1;
                                                                break;
                                                        }
                                                }

                if(rcv_num==ack_num){
                  ack_num=0;
                  printf("sliding window\n");
                 }
                   printf("WRITTING TO FILE:\n");
                                                 for(int k=1; k<=10;k++){
                                                        fwrite(pkta[k].data, 1, pkta[k].length, fptr);   /*Write the
recieved data to the file*/
                                                        printf("Packet.ID ---> %ld          Packet length written to
file---> %ld\n", pkta[k].ID,pkta[k].length);
                        bytes_rec += pkta[k].length;
                        pkt_test.length=pkta[k].length;
                        //if file received completely for k<10 we have to break the loop(when last packtes are
received)
                       if( pkta[k].length!=512){ //if last packets is received befor k=10
                         break;
                        } }



                    //10 packets are written in file
                /*************************************************/

                    //if all packets received
                                                  if (pktrecv == total_Packets) {
                                                    printf("File recieved\n");
                                                   }
                                        } //end of while loop


                                printf("Total Packets recieved ---> %ld\n",pktrecv );
            printf("Total Bytes recieved ---> %ld\n",  bytes_rec);
                                fclose(fptr);
                                }
                                else { //if server has sent number of packets to be received=0
                                        printf("File is empty\n");
                                }
            close(sockfd);
            exit(EXIT_SUCCESS);
return 0;
}
```

Server-Side Code:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include <dirent.h>
#include <stdbool.h>
#define PORT    8002
#define BUF_SIZE (512)                    //Max buffer size of the data in a frame

/*A packet with unique id, length and data*/
struct pkt {
            long int ID;
            long int length;
            char data[BUF_SIZE];
    long int ack;

};


int main(){

            struct sockaddr_in sv_addr, cl_addr;
            struct stat st;
            struct pkt pkt_test; //test pkt
            struct timeval t_out = {0, 0};
    struct pkt pkta[11];  //array of pkets to be received
            char   msg_recv[BUF_SIZE]; //buffer to received file name
            ssize_t numRead; //variable to store bytes of file name
            ssize_t length;//length of cl_addr
            off_t f_size;       //size of file to be sent
            int    ack_num = 1;  //Recieve file size and name packet acknowledgement
```

```
   int    total_pkts = 0;//total packtes in file
          int    sockfd; // sockt descriptor
   int    pktsnd=0;//sent pkets
   bool   check[11]={false}; //array for correct arrangement of pkts
   int    flag=1; // 10 pkts recveived flag
   int    drop_flag=0;
   int    resnd=0;
   int    rcv_num=0;//variable to store received acks
   FILE    *fptr;
/*****************************************************************************************
*****/

          /*Clear the server structure - 'sv_addr' and populate it with port and IP address*/
          memset(&sv_addr, 0, sizeof(sv_addr));
          sv_addr.sin_family = AF_INET;
          sv_addr.sin_port = htons(PORT);
          sv_addr.sin_addr.s_addr = INADDR_ANY;

         /**************************************************/


          if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
                    perror("Server socket creation failed\n");
   else{
      printf("Socket created:\n");
   }


          /**************************************************/
    //BINDING

          if (bind(sockfd, (struct sockaddr *) &sv_addr, sizeof(sv_addr)) == -1)
                    perror("Server bind failed:\n");
   else{
      printf("Binding done:\n");
   }
          /**************************************************/

   printf("Server: Waiting for client to connect\n");
          memset(msg_recv, 0, sizeof(msg_recv));

          /**************************************************/
   length = sizeof(cl_addr);
          if((numRead= recvfrom(sockfd, msg_recv, BUF_SIZE, 0, (struct sockaddr *) &cl_addr,
(socklen_t *) &length)) == -1)
                              perror("File name failed to receive:\n");
   else{
      printf("File name received:\n");
   }

          /**************************************************/
```

```
        //printf("Server: Recieved file name of %ld  from %s\n", numRead, cl_addr.sin_addr.s_addr);
        printf("Server: The recieved message ---> %s\n", msg_recv);
   printf("Server: Get called with file name --> %s\n", msg_recv);

   if (access(msg_recv, F_OK) == 0) {                          //Check if file exist

            int resend_frame = 0, drop_frame = 0, t_out_flag = 0;
                    stat(msg_recv, &st);
                    f_size = st.st_size;//Size of the file
                    t_out.tv_sec = 2;
                    t_out.tv_usec = 0;
                    setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, (char *)&t_out, sizeof(struct
timeval));   //Set timeout option for recvfrom

        /*************************************************/

                    fptr = fopen(msg_recv, "rb");       //open the file to be sent
                    if ((f_size % BUF_SIZE) != 0)
                      total_pkts = (f_size / BUF_SIZE) + 1;//Total number of packets to be sent
                    else
                              total_pkts = (f_size / BUF_SIZE);

        /*************************************************/

                    printf("Total number of packets ---> %d\n", total_pkts);
                    length = sizeof(cl_addr);
                    sendto(sockfd, &(total_pkts), sizeof(total_pkts), 0, (struct sockaddr *) &cl_addr,
sizeof(cl_addr));            //Send number of packets (to be transmitted) to reciever
                    recvfrom(sockfd, &(ack_num), sizeof(ack_num), 0, (struct sockaddr *) &cl_addr,
(socklen_t *) &length); //ack of sent number of pkts

                    while (ack_num != total_pkts)    {
                                        /*keep Retrying until the ack matches*/
                                        sendto(sockfd, &(total_pkts), sizeof(total_pkts), 0, (struct
sockaddr *) &cl_addr, sizeof(cl_addr));
                                        recvfrom(sockfd, &(ack_num), sizeof(ack_num), 0, (struct
sockaddr *) &cl_addr, (socklen_t *) &length);
                                        resend_frame++;

                                        /*Enable timeout flag even if it fails after 20 tries*/
                                        if (resend_frame == 20) {
                                                t_out_flag = 1;
                                                break;
                                        }
                     }


      /*when while loop starts
      /pktrecv is zero,total Packets are number of packets to  be sent
```

```
 */
/*transmit data packets sequentially followed by an acknowledgement matching*/
                while(total_pkts!=pktsnd)
                {
    rcv_num=0;
    ack_num=0;
    //if 10 acks has come then resend =0
    if(resnd==0){
       /**********************************************/

    //Read 10 pkets from file and store them in pkt array
    printf("READING FILE:\n");
    for(int k=1;k<=10;k++){
                          if((pkta[k].length = fread((pkta[k].data), 1, BUF_SIZE, fptr))!=0){
           printf("file bytes read --- %d\n",k);
           pkta[k].ID=k;//set ID of pkts
           pktsnd+=1;//total pkts sent yet
       }
       //for last pkts
       if(total_pkts==pktsnd){
            strncpy(pkta[k+1].data,"EOF",500);//copy end of file in last buffer
            printf("EOF copied\n");
            break;}
       if(pkta[k].length==0 && total_pkts!=pktsnd){
            printf("error reading file\n");
            exit(1);}
        }
       /**********************************************/


     //now send 10 pkts
     printf("SENDING TO CLIENT:\n");
     for (int k=1;k<=10;k++){
                         if((sendto(sockfd, &(pkta[k]), sizeof(pkta[k]), 0, (struct sockaddr *) &cl_addr,
sizeof(cl_addr)))>0){
           printf("Packets sent ----> %d  Packet ID%ld\n",k,pkta[k].ID);
           check[k]=false;//if pkt sent then tick its corrosponding entry in check array
        }             //send the Packet
       if((strncmp(pkta[k+1].data, "EOF",500)==0)){
           break;}

                         }}
       /**********************************************/
     //now receive 10 pkts acks
      printf("RECEIVINF ACKS:\n");
      for (int k=1;k<=10;k++){
          if((recvfrom(sockfd, &(pkta[k].ack), sizeof(pkta[k].ack), 0, (struct sockaddr *) &cl_addr,
(socklen_t *) &length))>0){      //Recieve the acknowledgement

            printf("Packet ----> %ld     Ack ----> %ld  Packet length ----> %ld
\n",pkta[k].ack,pkta[k].ack, pkta[k].length);
```

```
                    if( check[pkta[k].ack]!=true){   //if ticked entery is detected
                     check[pkta[k].ack]=true;
                     ack_num+=1;//sent pkt has acknowlged
                     }
                  }
                    if((strncmp(pkta[k+1].data, "EOF",500)==0)){//if next buffer is end of file
                       break;}
                    if(pkta[k].length!=512){
                       break;}
              }


                //check the array and see for pkts which are not acknowlged yet
                  for (int i=1;i<11;i++){
                      if(check[i]==false){
                       printf("flase flag\n");
                       drop_flag=1;
                       ack_num=resnd;
                       resnd=i;
                       }
                   }

                  if(resnd==0){
                     printf("No drop occured\n");
                  }
                  drop_flag=0;
                  if(resnd!=0){
                     printf("PKT LOSS%d\n",resnd);
                     exit(1);
                  }
                  drop_flag=0;
                  for (int i=1;i<11;i++){
                        check[i]==false;
                  }


                  sendto(sockfd, &(ack_num), sizeof(ack_num), 0, (struct sockaddr *) &cl_addr,
sizeof(cl_addr));            //Send number of packets (to be transmitted) to reciever
                                    recvfrom(sockfd, &(rcv_num), sizeof(rcv_num), 0, (struct sockaddr *)
&cl_addr, (socklen_t *) &length);
                  printf("ack number---> %d   recv number%d\n",ack_num,rcv_num);

                   //selective repeat
                   while (ack_num != rcv_num)  //Check for ack
                                              {
                                                      //keep retrying until the ack matches
                                                      sendto(sockfd, &(pkta[rcv_num].data),
sizeof(pkta[rcv_num].data), 0, (struct sockaddr *) &cl_addr, sizeof(cl_addr));
                                                      recvfrom(sockfd, &(ack_num), sizeof(ack_num), 0,
(struct sockaddr *) &cl_addr, (socklen_t *) &length);
                                                      printf("ack ---> %ld       dropped, %ld times\n");
```

```
                                        resend_frame++;

                                        printf("frame ---> %ld    dropped, %ld times\n");

                                        //Enable the timeout flag even if it fails after 200 tries
                                        if (resend_frame == 200) {
                                             t_out_flag = 1;
                                             break;
                                        }
                                   }

        if(rcv_num==ack_num){
           ack_num=0;
           printf("sliding window\n");
         }
                                   resend_frame = 0;
                                   drop_frame = 0;
                                   /*File transfer fails if timeout occurs*/
                                   if (t_out_flag == 1) {
                                        printf("File not sent\n");
                                        break;
                                   }

                              }
      printf("Total Bytes sent ---> %d\n",total_pkts);
      printf("sent file size %ld\n",f_size);

                              fclose(fptr);
                              t_out.tv_sec = 0;
                              t_out.tv_usec = 0;
                              setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, (char *)&t_out,
sizeof(struct timeval)); //Disable the timeout option
                         }
                         else {
                              printf("Invalid Filename\n");
                         }
     close(sockfd);
     exit(EXIT_SUCCESS);
     return 0;
          }
```