**Computer Networks**

**Project Report**

**“Reliable File Transfer Over UDP”**

Group Members

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

Contents

[Abstract: 3](#_Toc7980822)

[Advantages of using UDP: 3](#_Toc7980823)

[Disadvantages of using UDP: 3](#_Toc7980824)

[Key Features of Project: 3](#_Toc7980825)

[Process Diagram (Normal Flow): 3](#_Toc7980826)

[Finite State Machine FSM: 5](#_Toc7980827)

[Client-Side Output: 6](#_Toc7980828)

[Server-Side Output: 7](#_Toc7980829)

[Client-Side Code: 8](#_Toc7980830)

[Server-Side Code: 13](#_Toc7980831)

# 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):

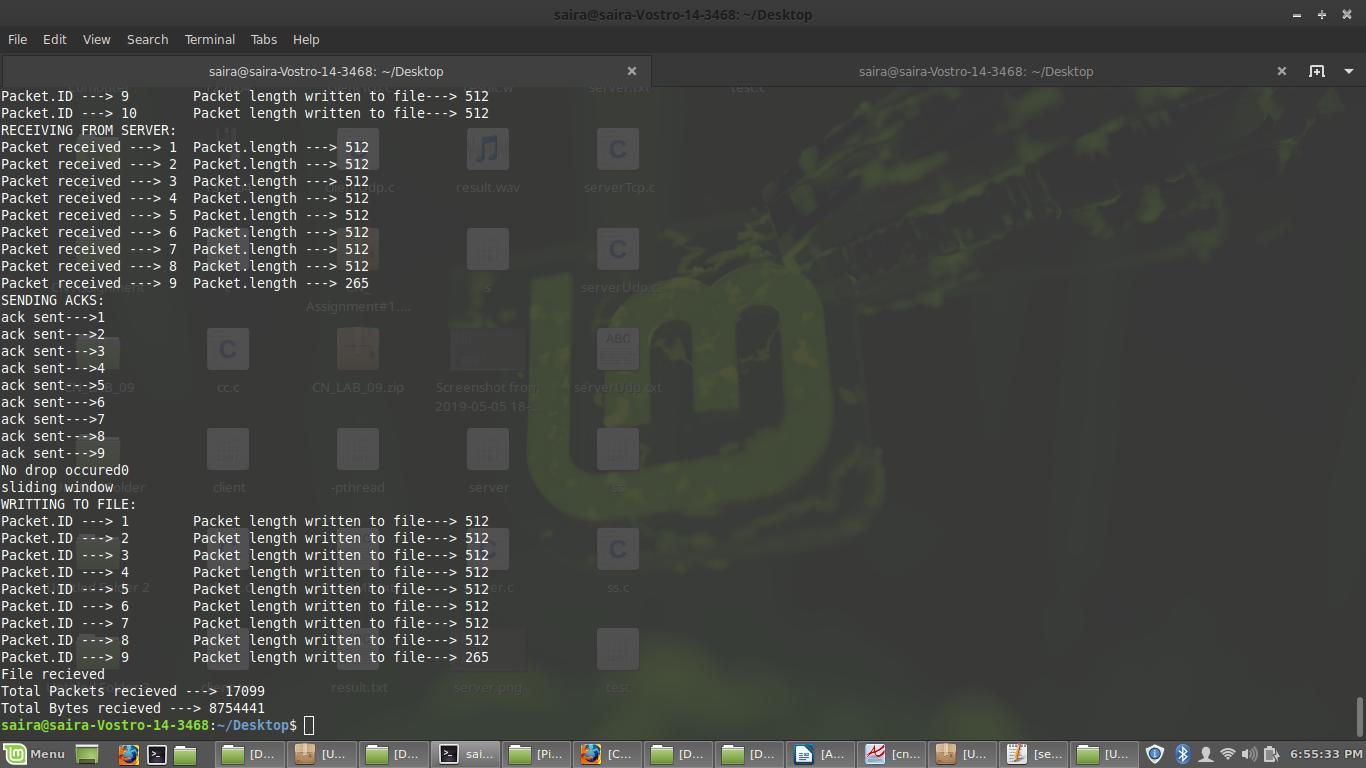


# Finite State Machine FSM:

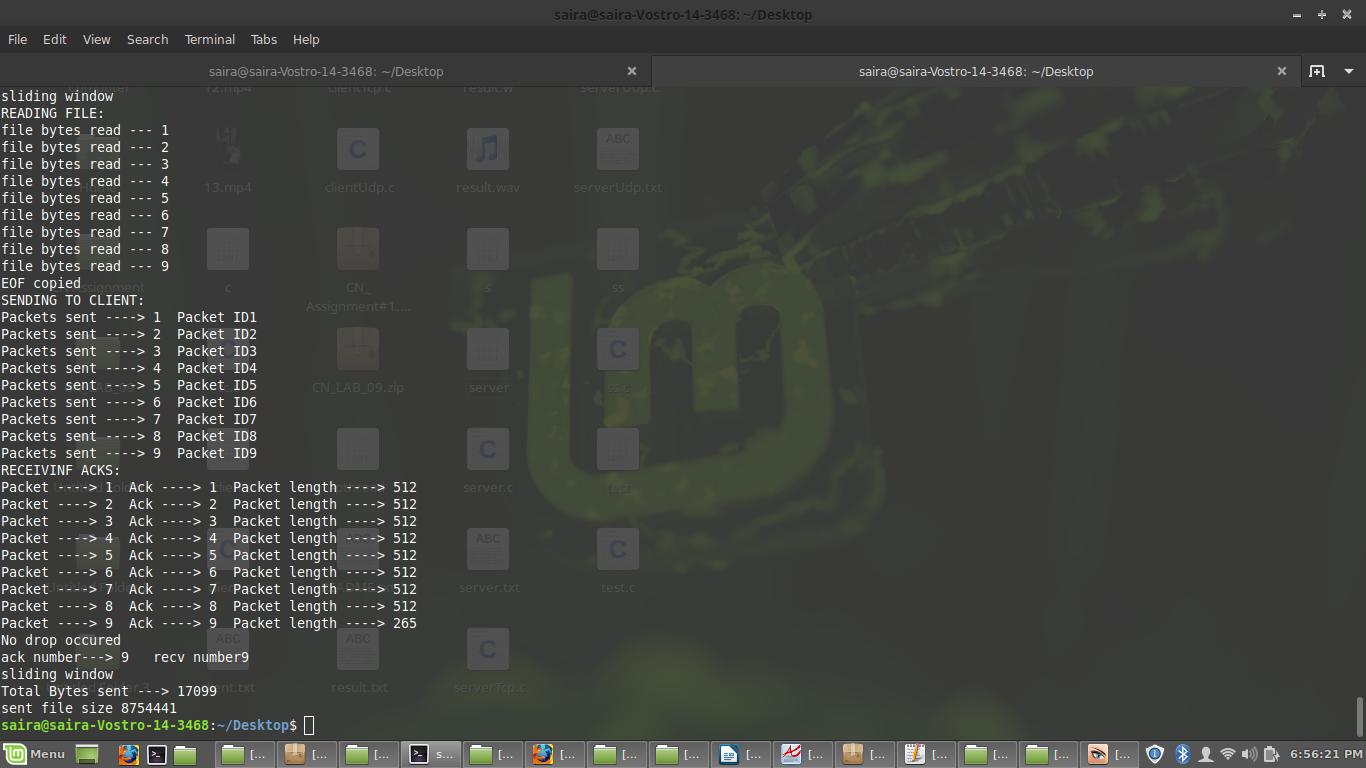


Note: For simplicity only seq # 0 and 1 is used here.

# Client-Side Output:



# Server-Side Output:



# 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;

}