Reliable File Transfer Using UDP

Presented By - Aniket Pyne

Working Principle: Client

1. Create Socket & connect to server.
2. Read the file and determine total no. of packets to be sent.
3. Send the details of the file name , amount of data per packet , total amount of data in the first calibration packet. And the actual data for the consequent packets.
4. Receive an acknowledgement.
    a. If it is the expected acknowledgement then send the next frame.
    b. Otherwise send the same frame again.
5. Send a connection termination packet at the end.

Working Principle: Server

1. Create Socket & bind to that socket.
2. Receive a packet from client.
   a. If it is the first frame then initialize the file information according to the received packet.
   b. If it is a data frame then put the frame data into the file buffer.
   c. If is a request to terminate connection then do so.
3. For every packet received if it the expected packet then keep it and send ack for next packet , otherwise send the same ack that was last sent.
4. Once packet transfer is complete then copy the file buffer into the output file created by server.

Frame Structure

No. of data bytes to be sent per frame

Array containing the data bytes

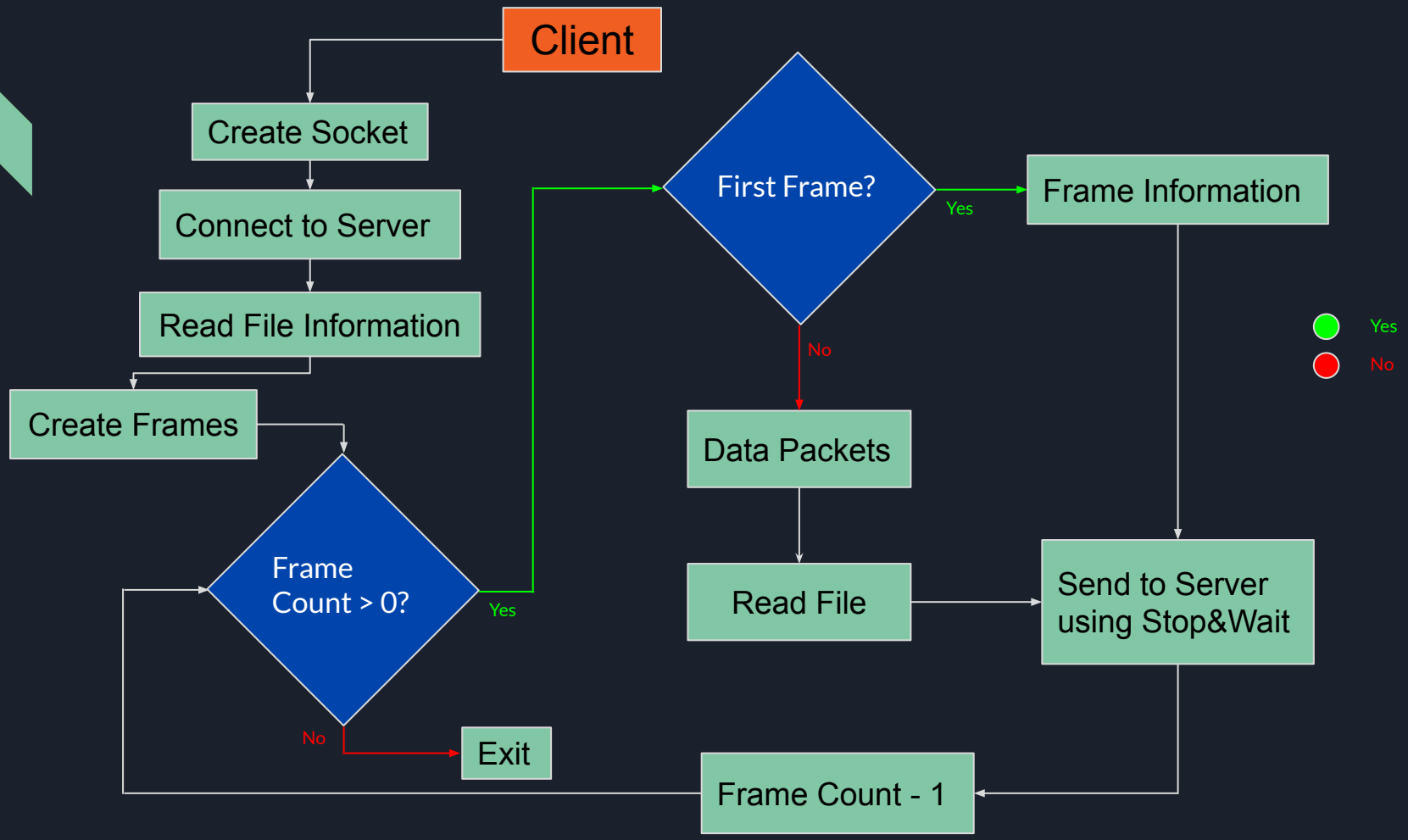
Array containing the file name

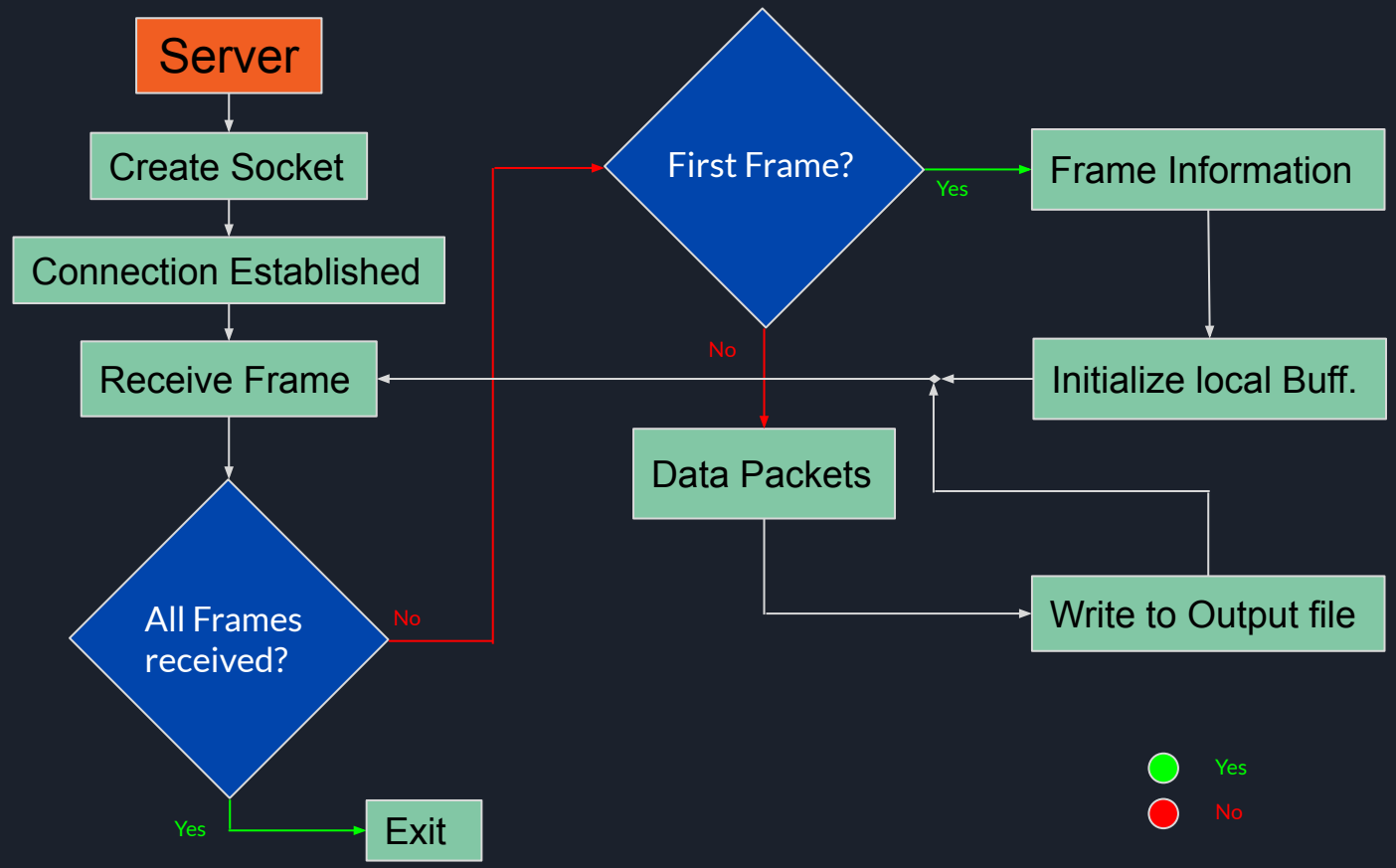
Frame type	Sqn No.	Ack No.	Buffer Size	Data [ ]	File name [250]	File Size
2 Bytes	2 Bytes	2 Bytes	2 Bytes	1212 Bytes	250 Bytes	2 Bytes

Total - 1472 Bytes

Types	Details
0	Sequence Frame/Data Frame
1	Terminate Connection Frame
2	Initial Calibration Frame

Flow Charts


Client
Create Socket
Connect to Server
Read File Information
Create Frames
Frame Count > 0?
Yes
No
Exit
First Frame?
Yes
No
Frame Information
Data Packets
Read File
Send to Server using Stop&Wait
Frame Count - 1
Yes
No


Server
Create Socket
Connection Established
Receive Frame
All Frames received?
No
Yes
Exit
First Frame?
Yes
No
Frame Information
Initialize local Buff.
Data Packets
Write to Output file
Yes
No

Implementation

Headers Used

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

#define data_per_packet 1212
```

Some functions used for file handling

fseek (fl, 0, SEEK_END)

File pointer

Offset

Index

fread (msg, 1, file_len, fl);

Buffer to put the data

size

No. of elements with size size to be copied

File pointer to input file

SEEK_END - End of file index

SEEK_SET - First index of file

ftell(fl) - returns the current index of the file pointer.

Some functions used for file handling

fwrite (msg, 1, file_len, fl);

Buffer to read the data from

size

No. of elements with size size to be copied

File pointer to the output file

fopen (file_name, mode);

Path of the file to be opened

Mode to open the file in.

r / rb	read / read binary
w / wb	write/ write binary
a / ab	append / append binary

One Important Code Segment

This code is used to send the correct slices of the input file per packet :

```
bytes_left = file_len - ((frm_no - 1) * buffer_size);
```

- Calculates how much bytes still need to be sent with considering the current frame.

```
fseek(fl, (-bytes_left) , SEEK_END);
```

- Sets the file pointer accordingly to the correct position.

```
(buffer_size < bytes_left)? fread(frm_send.data, 1, buffer_size,  fl) : fread(frm_send.data, 1, bytes_left,  fl);
```

- To avoid sending garbage bytes on the last frame. Explained more in subsequent slides.

Example - Client Side

Input File →

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7

No. of Frames - 4
(calibration & terminate connection frame not considered here)

Bytes_left = 7
Buffer_size = 2

Slice 0 Slice 1 .....

frm_send.data →

Byte 1	Byte 2

```
(buffer_size < bytes_left)? fread(frm_send.data, 1, buffer_size,  fl) : fread(frm_send.data, 1, bytes_left,  fl);
```

Input File →

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7

No. of Frames - 4

Bytes_left = 5
Buffer_size = 2

Slice 0

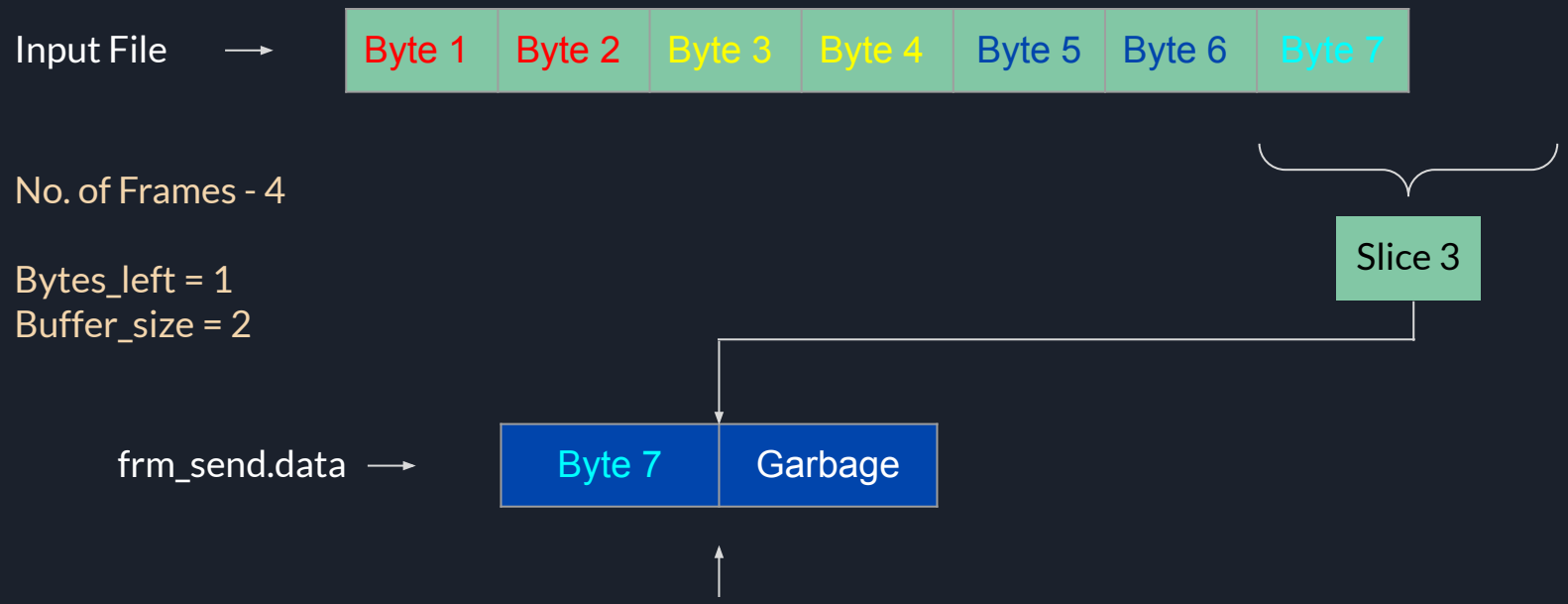
Slice 1

frm_send.data →

Byte 3	Byte 4

```
(buffer_size < bytes_left)? fread(frm_send.data, 1, buffer_size,  fl) : fread(frm_send.data, 1, bytes_left,  fl);
```

```
(buffer_size < bytes_left)? fread(frm_send.data, 1, buffer_size,  fl) : fread(frm_send.data, 1, bytes_left,  fl);
```



No. of Frames - 4

Bytes_left = 1
Buffer_size = 2

Bytes_left < Buffer_size

This condition tells upto which byte is actual file byte. This way only valid bytes will be read by fread().

Server Side

In calibration frame:

```
bytes_left = file_len; //bytes left to copy to file
```

On receiving each data frame :

```
(buffer_size < bytes_left)? fwrite(frm_recv.data, 1, buffer_size,  f2) : fwrite(frm_recv.data, 1, bytes_left,  f2);
bytes_left = bytes_left - buffer_size;
```

Bytes_left - contains how many bytes left to write to output file

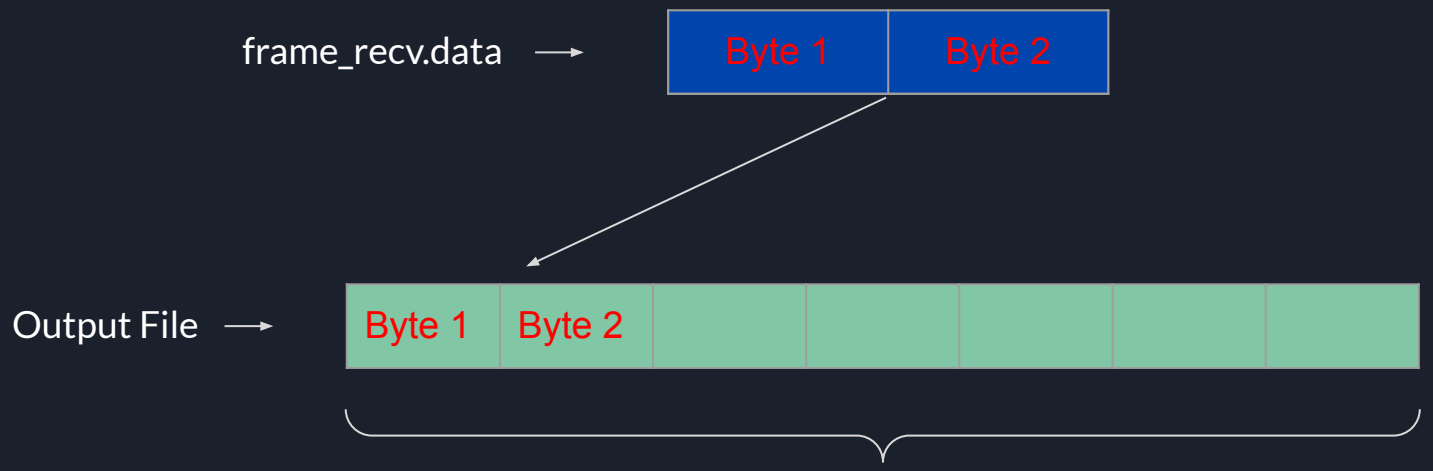
Buffer_size - no. of bytes per frame

F2 - output file pointer

First Frame :

```
(buffer_size < bytes_left)? fwrite(frm_recv.data, 1, buffer_size,  f2) : fwrite(frm_recv.data, 1, bytes_left,  f2);
bytes_left = bytes_left - buffer_size;
```

Initially bytes_left = 7 buffer_size = 2

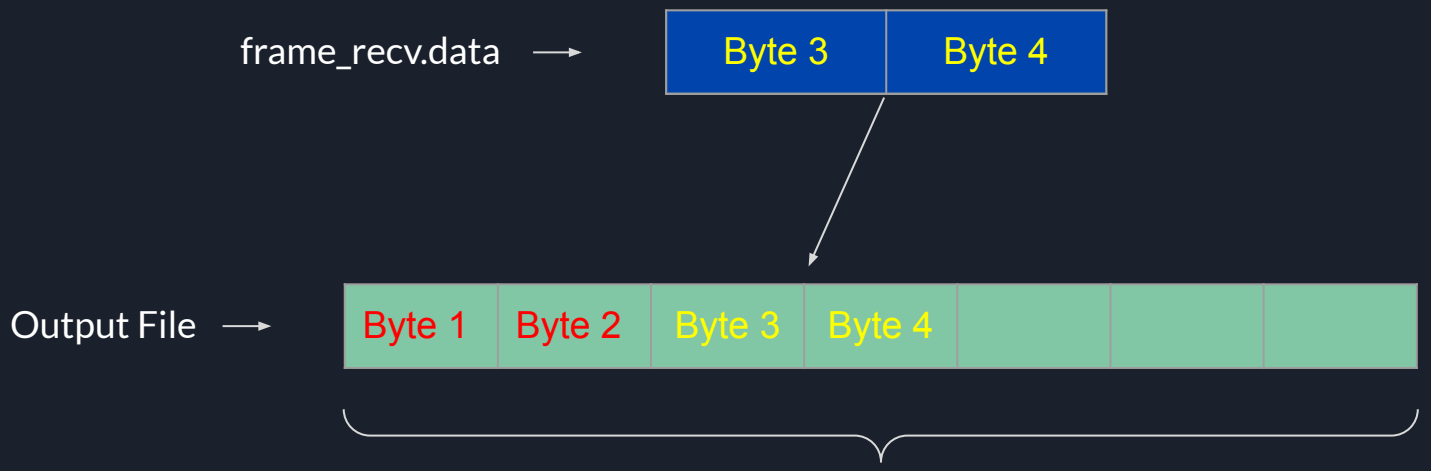


file_len = 7 - received from calibration frame

Second Frame :

```
(buffer_size < bytes_left)? fwrite(frm_recv.data, 1, buffer_size,  f2) : fwrite(frm_recv.data, 1, bytes_left,  f2);
bytes_left = bytes_left - buffer_size;
```

Initially bytes_left = 5 buffer_size = 2

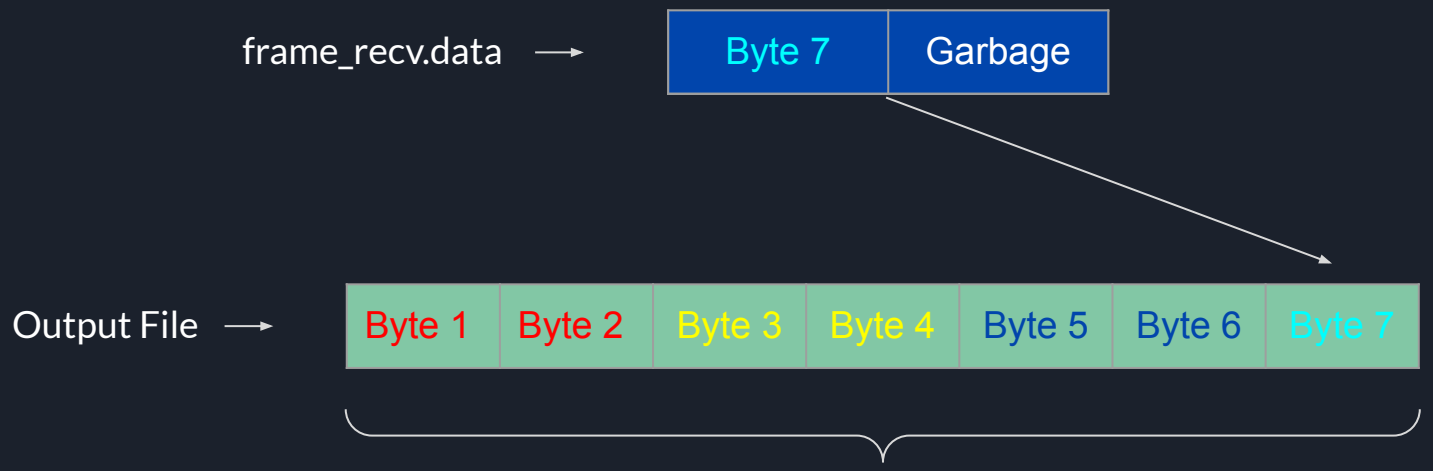


file_len = 7 - received from calibration frame

Last Frame :

```
(buffer_size < bytes_left)? fwrite(frm_recv.data, 1, buffer_size,  f2) : fwrite(frm_recv.data, 1, bytes_left,  f2);
bytes_left = bytes_left - buffer_size;
```

Initially bytes_left = 1 buffer_size = 2



file_len = 7 - received from calibration frame

Source Code Explanation

Client: Declaring File Transfer Parameters

```
long file_len; //file size
int buffer_size = data_per_packet; //data per packet
```

Client: The Frame Structure

```
//********************** Frame Structure
typedef struct frame{
    int frame_kind;     //0 - sqn/data frame, 1 - terminate connection, 2 - calibration
    int sqn_no;
    int ack;
    int buffer_size;
    char data[data_per_packet];         //Assuming only data_per_packet bytes of data can be transfered per frame.
    char file_name[250];
    int size_in_bytes;      // msg_len (file size in byte)
}Frame;
```

Client: Some Structures and data types used in select( )

```
fd_set rset;    // set of sockets structure to be monitor via select()
struct timeval tv;  //used for specifying the timeout period in the select() call
FD_ZERO(&rset);     // setting the set of sockets to 0
FD_SET(client_socket, &rset);   // putting our socket into the set to be monitored by select()
```

Client: The Socket Creation and Server Address Parameters Declaration

```
//********************************************************** Creating a Socket
int client_socket;
client_socket = socket(AF_INET, SOCK_DGRAM, 0);

struct sockaddr_in server_address;
server_address.sin_family = AF_INET;
server_address.sin_port = htons(5000);
server_address.sin_addr.s_addr = INADDR_ANY;
```

Client: Opening The File To be sent and getting its size

```
char file_name[250];
printf("Enter name of a file you wish to send\n");
gets(file_name);        //Get File name

FILE *fl = fopen(file_name, "rb"); //read bytes
fseek(fl, 0, SEEK_END); //file pointer goes to the end of the file
file_len = ftell(fl);       //ftell() returns the current address of the file pointer
fseek(fl, 0, SEEK_SET); //file pointer goes to the very beginning
printf("\nFile Size (Total No. of bytes) - %ld",file_len);
```

Client: Number of Frames Calculation

```
//*************************************************** Calculating No. of frames
float temp = (float)file_len / (float)buffer_size;
int frm_cnt = ceil(temp) + 1;
int frm_cnt_org = frm_cnt;
printf("\nNo. of frames %d\n" , frm_cnt);
```

Client: Initializing The Frame, Ack and Log Control Variables

```
int frm_no = 0;
Frame frm_send , frm_recv;

int ack_cnt = 0;     //count of recieved acks
int bytes_done = 0; //counter to keep track of total bytes sent
int timeouts = 0;    //No. of timeouts occured
int terminate_timeout = 0;  //this is counter of whether enough timeouts has occured
// for the client to stop trying to resend i.e connection is lost
int timeout_occured = 0; //for knowing if timeout has occured or not
int bytes_left ;
```

Client: Sending The Calibration (First) Frame

```
//*********************************************************************************** While runs till whole file is sent
while(frm_cnt > 0 ) // for all frames
{
    if(frm_no == 0) //first frame
    {
        frm_send.frame_kind = 2; //calibration frame
        frm_send.sqn_no = frm_no; //which no. frame it is

        frm_send.size_in_bytes =file_len;   //total file size
        frm_send.buffer_size = buffer_size; // packet data size

        strcpy(frm_send.file_name, file_name);  //file name is copied into the frame

        printf("Sending %d th frame\n" , frm_no);
        sendto(client_socket, &frm_send, sizeof(frm_send), 0 , (struct sockaddr *) &server_address, sizeof(server_address));
    }
```

Client: Sending The Data Frame

```
else //all other frames
{
    frm_send.frame_kind = 0; // syn or data frame
    frm_send.sqn_no = frm_no; //which no. frame it is

    bytes_left = file_len - ((frm_no - 1) * buffer_size);       // how much bytes left to be sent while sending current frame
    fseek(fl, (-bytes_left) , SEEK_END);        // set the file pointer to the current postion accordingly
    (buffer_size < bytes_left)? fread(frm_send.data, 1, buffer_size,  fl) : fread(frm_send.data, 1, bytes_left,  fl);
    // to make sure only proper file bytes are sent and not any garbage.

    printf("\nSending %d th  frame\n" , frm_no);
    sendto(client_socket, &frm_send, sizeof(frm_send), 0 , (struct sockaddr *) &server_address, sizeof(server_address));

    bytes_done += buffer_size;  //how much total bytes sent
}
```

Client: Timer Implementation and timeout check

```
tv.tv_sec = 2;  //sec
tv.tv_usec = 0; //usec
FD_ZERO(&rset);     // setting the set of sockets to 0
FD_SET(client_socket, &rset);   // putting our socket into the set to be monitored by select()
select(client_socket + 1, &rset, NULL, NULL, &tv);  // wait here till some packet is recieved
// at the socket or 2s (as per value in tv) has passed
if (FD_ISSET(client_socket, &rset) == 0)    // no data available to read
{
    timeout_occured = 1;
}
else    // there is data available to read
{
    // receive ack from server
    recvfrom(client_socket, &frm_recv, sizeof(frm_recv), MSG_WAITALL, NULL, NULL);  //recieve ack
}
```

Client: Received ack check for validity as well handling resume request

```
    if (timeout_occured == 0 && frm_recv.ack == frm_no + 1) //Check if the valid ack is recived or not while no timeout
    {
        printf("\nRecieved ack - %d", frm_recv.ack);
        ack_cnt = frm_recv.ack; //correct ack was recieved
        terminate_timeout = 0;  //this is resetted here as a valid ack suggest that connection is still there.

        frm_cnt = frm_cnt_org - (frm_recv.ack);  // one frame is succesfully sent
    }
    else if (timeout_occured == 1)  // timeout has occured
    {
        printf("\nRecieved invalid ack which is - %d....Resending" , frm_recv.ack);
        timeout_occured = 0;    // reset the timeout flag
        bytes_done -= buffer_size; //as the last packet was not succesfully sent
        timeouts++; //total timeouts
        terminate_timeout++;
        if(terminate_timeout > 25)
        {
            printf("\nToo many timeouts....connection lost");
            break;
        }
        frm_recv.ack = frm_no; // this line is to reset back to the correct ack from the
        //garbage one which will occur when recvfrom() doesnt receives anything
    }
    else if(frm_cnt_org - (frm_recv.ack) <= 0)  // this is used when the transfer is already complete.
    // then the server will send a ack which will be equal to the total no. of frames.
    {
        printf("\nTransfer already Complete");
        close(client_socket);
        printf("\nClient socket closed ");

        return 0;
    }
    else
        printf("\nReceived acknowledgement %d for resuming transfer", frm_recv.ack);
    frm_no = frm_recv.ack;   // next frame to be sent

    printf("\tPercentage done- %f ", ((float)(frm_cnt_org - frm_cnt)  * 100)/frm_cnt_org);
} // end of while
```

Client: Creating and Populating the Log File and closing the input file

```
fclose(fl);     // Close the input file

//********************************************** Calculating Some Statistics
float packet_loss = 100 - (((float)ack_cnt *100)/frm_cnt_org);
printf("\nPacket loss - %f\n", packet_loss);
printf("\nTimeouts - %d\n", timeouts);

//********************************************** Writing to a log file
FILE *f4 = fopen("log_udp.txt", "w");
fputs("Packet loss - ",  f4);
fprintf(f4, "%f", packet_loss);

fputs("\nTimeouts - ",  f4);
fprintf(f4, "%d", timeouts);
fclose(f4);     //Close the log file
```

Client: Connection Termination and Closing The Socket

```
//****************************** Sending a frame for Close connection request
frm_send.frame_kind = 1; //termination request frame
frm_send.sqn_no = frm_no;
sendto(client_socket, &frm_send, sizeof(frm_send), 0 , (struct sockaddr *) &server_address, sizeof(server_address));
//*****************************************************************************************************************
close(client_socket);
printf("Client socket closed \n");
```

Server: Declaring The Frame Structure

```
//********************** Frame Structure
typedef struct frame{
    int frame_kind;
    int sqn_no;
    int ack;
    int buffer_size;
    char data[data_per_packet];
    char file_name[250];
    int size_in_bytes;
}Frame;
```

Server: Some Structures and data types used in select( )

```
fd_set rset;    // set of sockets structure to be monitor via select()
struct timeval tv;  //used for specifying the timeout period in the select() call
FD_ZERO(&rset);     // setting the set of sockets to 0
FD_SET(client_socket, &rset);   // putting our socket into the set to be monitored by select()
```

Server: Preparing The Server Socket

```
//****************************************** Socket Creation
int server_socket ;
server_socket = socket(AF_INET, SOCK_DGRAM, 0);

struct sockaddr_in server_address , client_address;
server_address.sin_family = AF_INET;
server_address.sin_port = htons(5000);
server_address.sin_addr.s_addr = INADDR_ANY;

bind(server_socket, (struct sockaddr *) &server_address, sizeof(server_address));
int cli_len = sizeof(client_address);
```

Server: Declaring File and Frame information Variables and arrays for handling resume op.

```
int i=0, j=0; //some loop variables
Frame frm_send , frm_recv;

long file_len; //no. of bytes in the file
int buffer_size; //size of data (in bytes) for each frame
int bytes_left; //bytes left to copy to file
int expected_frm_no = 0;
char file_name[200];

FILE *f2;   //a file pointer to the output file
FILE *f5;   //a file pointer to the log file

frm_send.ack = 1;   // first ack will be 1 for the 0th frame

char log_file_name[200];    //array containing the name of the currently open log file
char currently_open_file[200];  //array containing the name of the file currently being
//transfered ( this will never contain the name of a log file)***
```

Server: Implementing timer and receiving msg from client

```
while(1)    // main loop for receiving frames
{
    //************************************** implementing timer functionalities
    tv.tv_sec = 10; //sec
    tv.tv_usec = 0; //usec
    select(server_socket + 1, &rset, NULL, NULL, &tv);  // wait here till some packet is recieved
    // at the socket or 10s (as per value in tv) has passed
    if (FD_ISSET(server_socket, &rset) == 0)
    {
        printf("\nNo Message for too long......Server Exiting");
        return 0;
    }
    //*************************************************************************

    //recieve frame from client
    recvfrom(server_socket, &frm_recv, sizeof(frm_recv), MSG_WAITALL,( struct sockaddr *) &client_address,&cli_len);
```

Server: Calibration Frame reception

File information Variables are assigned according to received frame:

```
//Work on the recived frame
if(frm_recv.frame_kind == 2 )//calibration frame
{
    //****************************************Receive the information of the file to be transfered

    file_len = frm_recv.size_in_bytes; // get total file size

    buffer_size = frm_recv.buffer_size;  // get no. of bytes per frame
    strcpy(file_name,frm_recv.file_name); //store the recieved file name
    //*****************************************************************************************************

    strcpy(log_file_name,frm_recv.file_name);
    strcat(log_file_name, "_resume_log.txt");   // generate the file specific log file name which will be used later
    //to check if a log exists or not.
```

Server: Check for any open file and closing them as necessary

```
if (strcmp(currently_open_file,file_name) != 0)     // checks if the file to be tranfered that is received from the client is
                                                    //already open or not. If yes to avoid reopening which will cause errors.
{
    // this 'if' is used to close any already open file as they will be opened in subsequent lines.
    // if no file is open then len(currently_open_file) will be 1 and we wont have to close any file.
    // Doing this is to make sure whatever was written to the file before is actually properly saved.
    if(strlen(currently_open_file) >= 2)    // if any file is open then length of currently_open_file will have to be
                                            //more than 2 ( . in 'file_name.extension' and the '\0' char is only considered here)
    {
        fclose(f2);
        fclose(f5);
    }
    bytes_left = file_len; //bytes left to copy to file
    // this is here considering we are having the file first time. if any log exists then
    // it will be updated accordingly

    // LOG HANDLER (Explained later)
}
else // when the connection was restarted for the same file but server never stopped.
    expected_frm_no-- ; //So this calibration should not change this variable,so we
                        //decrement it here to compensate for the increment at the last line of loop;
```

Server: LOG HANDLER - No log file present

```
if ((f5 = fopen(log_file_name, "r")) == NULL)
// if no log file exists  then it is a new file which will be transfered from beginning
{
    f5 = fopen(log_file_name, "w");
    // We will open a new log file for storing transfer percentage information
    f2 = fopen(file_name, "wb");    //open the output file with the file name
    strcpy(currently_open_file,frm_recv.file_name); //update this currently_open_file accordingly

    expected_frm_no = 0;    // as we are considering from the very beginning

    //********************************************************************** write to log file
    fputs(file_name,  f5);          // 1st line of log file
    fprintf(f5, "\n%d", buffer_size);   // 2nd line of log file
    fprintf(f5, "\n%d", frm_recv.sqn_no);   //for keeping track of progress.
    fprintf(f5, "\n%d", bytes_left);
    //*********************************************************************************************
    frm_send.ack = frm_recv.sqn_no + 1; // Preping the ack packet
}
```

Server: LOG HANDLER - log file present

```c
else
{
    printf("\nLog file found");        //resume transfer according to the log file. File is already open in the if statement.

    fgets(file_name, sizeof(file_name), f5);        // read file name from log
    file_name[strcspn(file_name, "\n")] = 0;    // discarding the newline character

    char temp_buffer_size[100];
    fgets(temp_buffer_size, sizeof(temp_buffer_size), f5);      // get the buffer_size in char format
    buffer_size = atoi(temp_buffer_size);       // convet char buffers_size to int

    f2 = fopen(file_name, "ab");        // Open the partially transfered file

    strcpy(currently_open_file,file_name);  //update this currently_open_file accordingly

    printf("\t%s", file_name);

    char line[100] , last_line[100], last_to_last_line[100];
    while (fgets(line, sizeof(line), f5))       // extract the last recieved frame and how much bytes left to be tranfered
    {                                           // from the log file which is in the last two line respectively
        strcpy(last_to_last_line,last_line);
        strcpy(last_line,line);
    }

    int last_frame = atoi(last_to_last_line);   //convet char to int & set the control variables accordingly
    frm_send.ack = last_frame + 1;
    expected_frm_no = (last_frame + 1) - 1;     // the -1 at the end is of offsetting the expected_frm_no++ at the last line of the loop
    bytes_left = atoi(last_line);

    fclose(f5);
    if(bytes_left <= 0) // server determines that the whole file is already transfered from the log file
    {
        printf("\nFile fully transfered already");
        fclose(f2);
        sendto(server_socket, &frm_send, sizeof(frm_send), 0 , (struct sockaddr *) &client_address, sizeof(client_address));    //send last ack
        printf("\nSending ack - %d to tell client to stop",frm_send.ack );
        close(server_socket);
        printf("\nServer socket closed \n ");
        return 0;
    }
    f5 = fopen(log_file_name, "a"); // open the log file again in apend mode to continue writing subsequent frames
```

Server: Log file Structure

Line no .1 -	File Name	
Line no .2 -	Buffer Size	
Line no . 3 + (i) -	i for frame i.	i starts from 0
Line no . 3 + (i +1) -	bytes left after receiving frame i	

Repeated for all the received frames

Server: Data Frame Reception - Copying the File data from the received packet into the Output File

```
else if(frm_recv.frame_kind == 0 && frm_recv.sqn_no == expected_frm_no) //data frame
{
    (buffer_size < bytes_left)? fwrite(frm_recv.data, 1, buffer_size,  f2) : fwrite(frm_recv.data, 1, bytes_left,  f2);
    bytes_left = bytes_left - buffer_size;   // how much bytes left after recieving this frame

    //************************* write to log file
    fprintf(f5, "\n%d", frm_recv.sqn_no);   //for keeping track of progress.
    fprintf(f5, "\n%d", bytes_left);
    //************************

    frm_send.ack = frm_recv.sqn_no + 1; // Preping the ack packet
}
```

Server: Connection Termination

```
else if(frm_recv.frame_kind == 1 && frm_recv.sqn_no == expected_frm_no) // termination frame
{                                       // ^ this condition is for preventing the server from stoping when there is still transfer
                                        //   remaining even if client tells to stop.
    printf("\nRecieved req to terminate connection\n");
    break;
}
else    // when expected frame is not recieved.
    expected_frm_no--; //to have the correct expected frame as this variable will be incremented at the last line of loop always.
```

Server: Sending the acknowledgement to client for each received frame

```
    sendto(server_socket, &frm_send, sizeof(frm_send), 0 , (struct sockaddr *) &client_address, sizeof(client_address));
    printf("\n Sent ack - %d\n",frm_send.ack );

    expected_frm_no++;
} // end of while
```

Server: Closing the output & log file & the socket

```
fclose(f2); // close the output file
fclose(f5); // close the log file

//*****************************************************************
close(server_socket);
printf("\nServer socket closed \n ");
```

Thanks