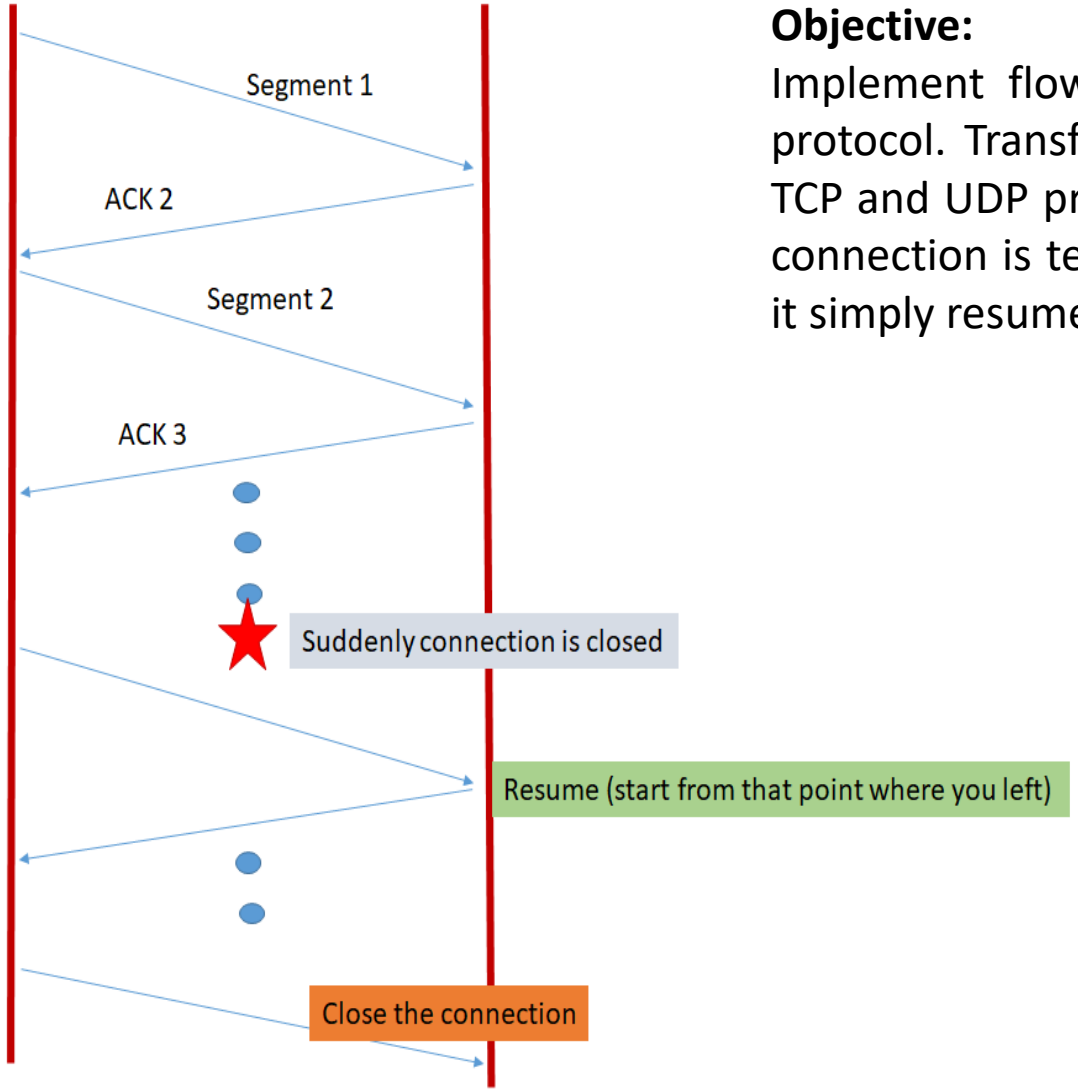
CSS1051: Networking Lab

ASSIGNMENT 3

Topic: Flow Control Implementation (Stop & Wait) using TCP & UDP protocol

PROBLEM STATEMENT



Objective:

Implement flow control mechanism using stop & wait protocol. Transfer files (Text, Image, Audio, Video) using TCP and UDP protocol. If during the connection suddenly connection is terminated then you have start ones again, it simply resume the process not start from being.

PROBLEM STATEMENT

Implementation Instructions:

Write a socket program in C/Java for Multimodal File Transmission using TCP and UDP with Full-Duplex Stop and Wait protocol. The program/protocol should support the following properties/mechanism

1. The protocol will send any type of files
2. Each packet should consist of the file name, sequence number/Acknowledgement number
3. A log file should be generated with some information like,

List of uncommon files in server and client which are to be transferred, Start time, If the connection is broken then the % of the file already uploaded, How many times connections were established during the complete transmission, End time (when the file is fully transmitted), How many packets are lost, How many time-outs are occurred, etc.

PROBLEMS ADDRESSED

1. The program can transfer any type of file
2. A log file is generated to keep track of different information like current segment number, transfer starting time etc.
3. Manual re-transmission for broken connection under TCP protocol.

WHY AGAIN TCP IN STOP & WAIT

WHY AGAIN TCP IN STOP & WAIT

As we know already TCP in default wait for response after sending a message to other party. Though retransmission of stop & wait protocol is not implemented in TCP by default that's why implementation of stop & wait protocol through TCP has a proper significance.

PREREQUESTIES

PREREQUESTIES

To implement the program of TCP Stop & Wait transmission, we need to first resolve two unique challenges

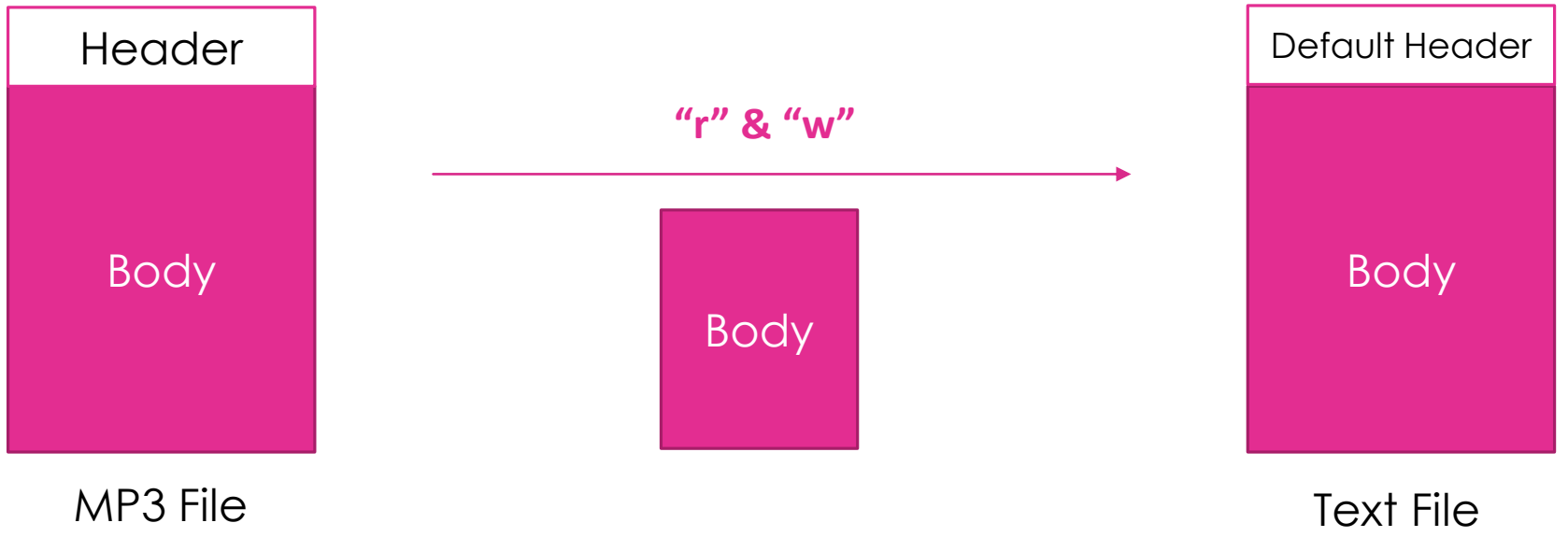
1. *File Read & Write*
2. *TCP Socket Communication (Client & Server)*
3. *Stop & Wait Implementation*

Now depend upon the programming language, pre-defined libraries, functions and other parameters will changed. Here "**C language**" is used to implement the TCP Stop & Wait protocol.

BINARY FILE R&W

WHY BINARY FILE R&W

If you are using only read mode **“r”** then header part is replaced with a default data by the compiler. But as we need to transfer any type of file so the header part will vary for each type of file. So here we have to use read binary mode **“rb”** to get access in the header portion also.



FILE READ & WRITE

Following functions were used to complete the objectives given in problem statement

1. fopen

file = fopen(filename, "rb");

Filename

Mode to open the file

2. fseek

fseek(file, 0, SEEK_END);

Position of pointer in file

fseek is used to seek to the desired position in file

Offset from the position

3. ftell

long fsize = ftell(file);

In case of SEEK_END, position of file pointer is unknown, therefore ftell is used.

File pointer

4. fclose

fclose(file);

File pointer

BINARY FILE READ & WRITE (CONTD.)

Following functions were used to complete the objectives given in problem statement

1. fread

fread(&temp[j],sizeof(char), 1, file);

- Address of memory buffer to store data
- Size in bytes of each element to be read
- File pointer of file to be read
- Number of elements to be read

2. fwrite

fwrite(&tInfo, sizeof(tInfo), 1, fp);

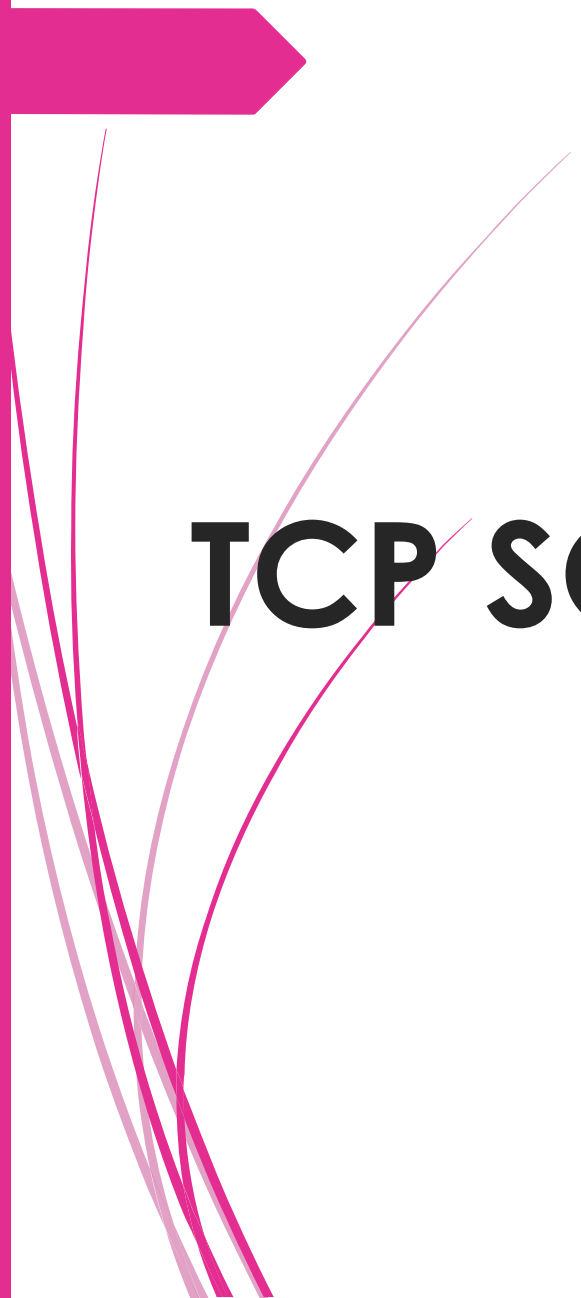
- Address of memory buffer to write data
- File pointer of file to be written

EXAMPLE CODE SNIPPET 1 (R/W)

```
FILE *file;
file = fopen(filename, "rb"); // File opened to read in binary mode
fseek(file, 0, SEEK_END);
long fsize = ftell(file);
fseek(file, 0, SEEK_SET);
```

SEEK TO THE END OF FILE

SEEK TO THE BEGINNING



TCP SOCKET COMMUNICATION

SOCKET :TCP CLIENT SIDE

```
// This following code snippet will help to connect to server
host = gethostbyname("127.0.0.1");

if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("Socket");
    exit(1);
}

server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(PORT);
server_addr.sin_addr = *((struct in_addr *)host->h_addr);
bzero(&(server_addr.sin_zero),8);

if (connect(sock, (struct sockaddr *)&server_addr,
            sizeof(struct sockaddr)) == -1)
{
    perror("Connect");
    exit(1);
}
```

SOCKET :TCP SERVER SIDE

```
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("Socket");
    exit(1);
}
if (setsockopt(sock,SOL_SOCKET,SO_REUSEADDR,&true,sizeof(int)) == -1) {
    perror("Setsockopt");
    exit(1);
}
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(PORT);
server_addr.sin_addr.s_addr = INADDR_ANY;
bzero(&(server_addr.sin_zero),8);

if (bind(sock, (struct sockaddr *)&server_addr, sizeof(struct sockaddr)) == -1) {
    perror("Unable to bind");
    exit(1);
}
if (listen(sock, 5) == -1) {
    perror("Listen");
    exit(1);
}

printf("\nTCPServer Waiting for client on port 5000");
fflush(stdout);
strcpy(tInfo.messageID,"");
```

STOP & WAIT IMPLEMENTATION

STOP & WAIT (CLIENT SIDE)

```
for(i=0,j=0; i < (partitions); i++,j++) { // File reading byte by byte and store in buffer for transmission
    if(j==bufferSize){
        send(sock,temp,bufferSize,0); // Send buffer data to server
        bytes_recieved=recv(sock,recv_data,1024,0);  // Wait to receive response from server
        recv_data[bytes_recieved] = '\0';
        if(strcmp(recv_data,"received")==0){
            tInfo.noSegments+=1;
            fp = fopen("logClient", "wb");
            fwrite(&tInfo, sizeof(tInfo), 1, fp);
            fclose(fp);
            // sleep(1);
        }
        printf("\nRecieved segment = %d\n" , tInfo.cS.segmentNo);
        // fflush(stdout);
        j=0; // Reset buffer

    }
    fread(&temp[j],sizeof(char), 1, file); // Overwrite/New write to buffer
    tInfo.cS.segmentNo=i;
    if((i+temoSegmentNo)==fsize){
        break;
    }
}
```

Updating the log after each transfer acknowledgement

STOP & WAIT (SERVER SIDE)

```
}else if(transferSequence==2){
    printf("==== CHECKPOINT 6 ====\n");
    printf("Offset string = %s\n",recv_data);
    tInfo.cS.segmentNo=atoi(recv_data);
    printf("Offset integer = %s\n",recv_data);
    send(connected, "offset",strlen("offset"), 0);
    if(tInfo.cS.segmentNo==0){
        printf("File reset\n");
        fp = fopen(tInfo.messageID, "wb");fclose(fp);
    }
    transferSequence=0;
    readOffset=tInfo.cS.segmentNo;
```

Receiving the offset from client

File reset for new transfer

WORKING PRINCIPLE

WORKING PRINCIPLE (CLIENT)

1. Define **Packet Structure.**
2. **Read file** which is to be sent and divide them in sections.
3. Define a **socket connection** using TCP and connect to the server.
4. Create an empty log file if not available.
5. **Stop and Wait protocol** to send files

   Loop until all segments are sent:

   a. Add message, sq.no., and flag as SQ to the frame.

   b. Send message and store it in log file.

   c. If **acknowledgement received**: add in log file and continue.

   d. Else: resend the same frame.

WORKING PRINCIPLE (SERVER)

1. Define **Packet Structure**.
2. Create a **socket** using TCP and wait for the client to connect.
3. Create an empty log file.
4. Create an empty file to save the packets.
5. **Stop and Wait Protocol:**

   Loop until all frames are received:

   a. **Receive message**

   b. Check if this is a repeated message,

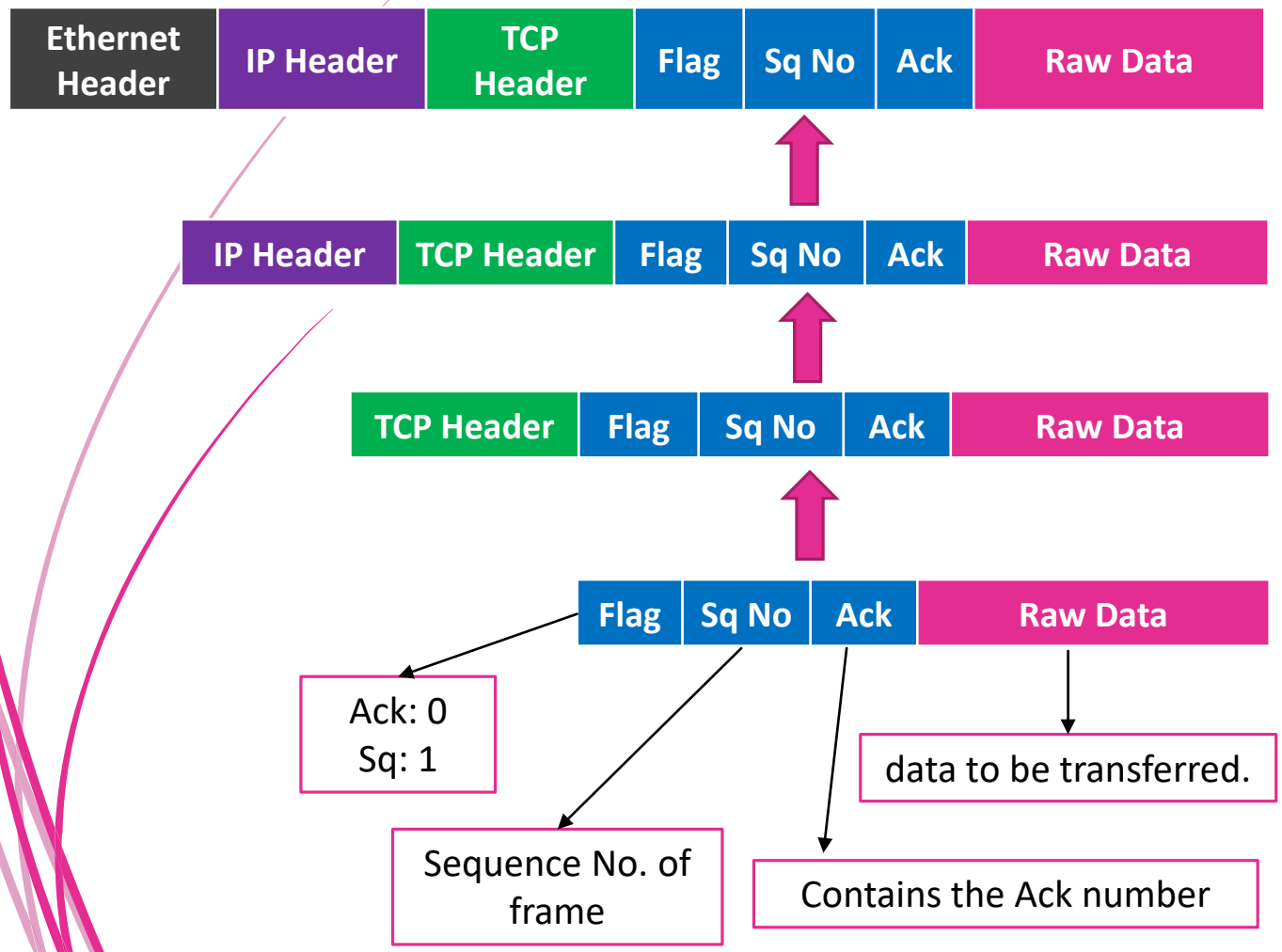
   c. if not:

      a. **write** it on the file
      b. store progress in **log** file.

   d. Add Acknowledgment. No. , and flag as ACK to the frame.

   e. Add in log file and continue.

PACKET STRUCTURE



Create header using a structure:

- **Flags:** Used to indicate the type of packet. The packet can be a sequence packet, acknowledgement packet.
- **Sequence number:** Indicates the frame sequence number which is sent to the server.
- **Acknowledgement Number:** Indicates the acknowledgement for its frame.
- Can add more information if required.

PACKET STRUCTURE (IMPLEMENTATION)

```
struct transferInfo{
    char messageID[100];                    // Message
    int noSegments;                         // Total No. of Segments
    float fileSize;                         // file Size
    time_t transferStartedAt; // file transfer starting time
    time_t transferCompletedAt; // file transfer completed time
    struct currentState{
        float timeElapsed; // total time taken to transfer
        int segmentNo; // Current packet no.
        int transferCompleted; // 0 = not completed / 1 = completed
    }cS;
}tInfo;
```

WHEN TO SEND THE PACKET STRUCTURE

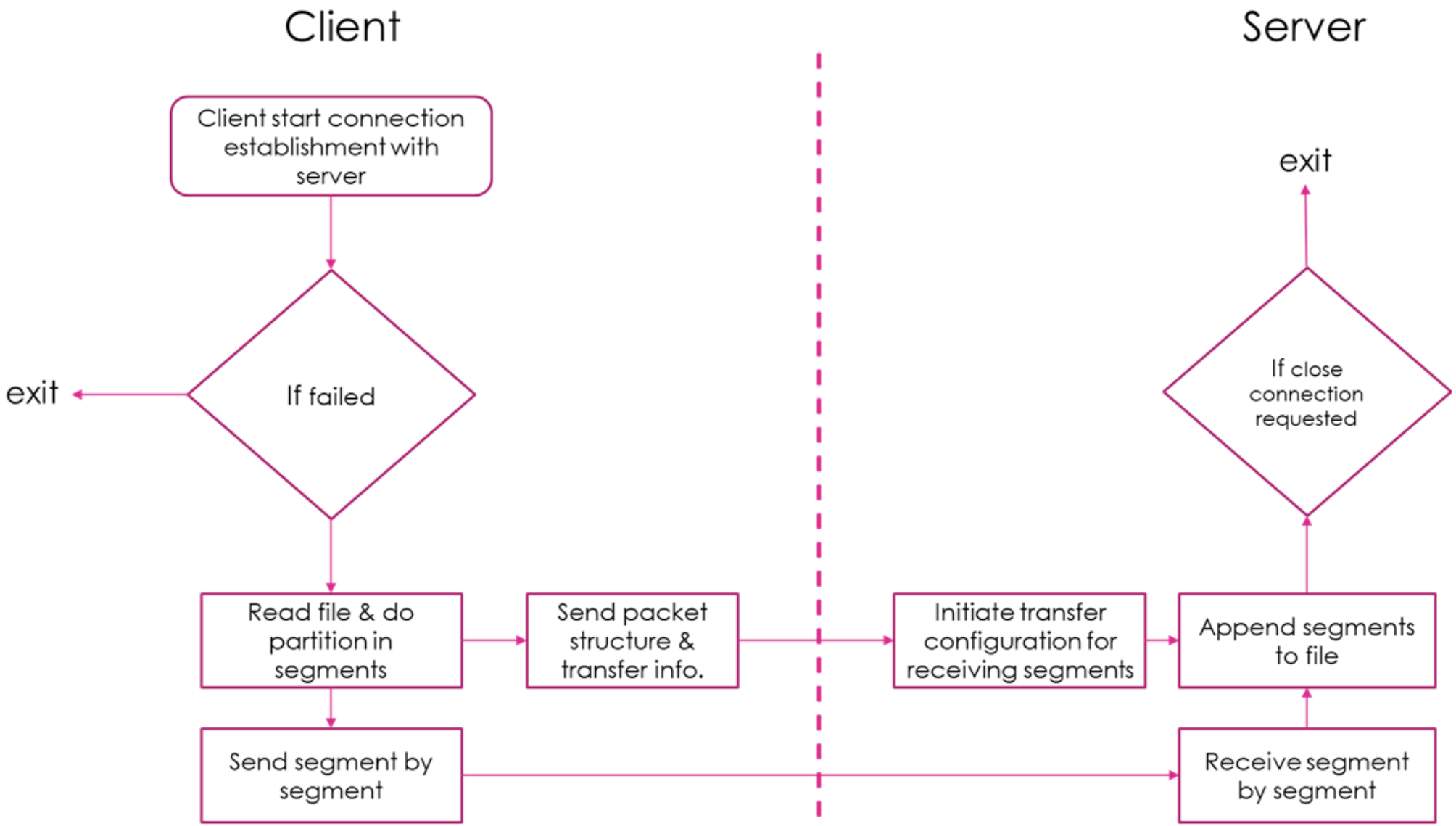
1. Concurrent server

Here the packet structure with file configuration details & data need to be sent every time.

2. Sequential server

In case of sequential server the packet structure with file data need to be sent every time. But file configuration can be sent once in time beginning the transfer.

PROPOSED ARCHITECTURE



IMPLEMENTATION

HEADER FILES

```
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>
#include <errno.h>

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

#include <time.h>
```

C Socket libraries

C Standard libraries

C Time handling library

PACKET STRUCTURE

```
#define bufferSize 10
#define PORT 5000
```

Fixed segment size for each transfer (Changeable)

Server PORT

```
struct transferInfo{
    char messageID[100]; //fileName
    int noSegments; // No of segments required to completely transfer the file
    float fileSize; // Filesize calculation
    time_t transferStartedAt; // Transmission starting time
    time_t transferCompletedAt; // Transmission ending time
    struct currentState{
        float timeElapsed; // Total time taken to transfer
        int segmentNo; // calculated as partition
        int transferCompleted; // 0 = not completed / 1 = completed
    }cS;
}tInfo;
```

Structured data
[Transmission logging]

FILE INPUT FROM COMMAND LINE

```
int main(int argc,char* argv[]){
    if(argc<2){
        printf("Enter filename\n");
        return 0;
    }
```

CLIENT-SERVER SOCKET ESTABLISHMENT

CLIENT SIDE

```
// This following code snippet will help to connect to server
host = gethostbyname("127.0.0.1");

if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("Socket");
    exit(1);
}

server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(PORT);
server_addr.sin_addr = *((struct in_addr *)host->h_addr);
bzero(&(server_addr.sin_zero),8);

if (connect(sock, (struct sockaddr *)&server_addr,
            sizeof(struct sockaddr)) == -1)
{
    perror("Connect");
    exit(1);
}
```

SERVER SIDE

```
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("Socket");
    exit(1);
}
if (setsockopt(sock,SOL_SOCKET,SO_REUSEADDR,&true,sizeof(int)) == -1) {
    perror("Setsockopt");
    exit(1);
}
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(PORT);
server_addr.sin_addr.s_addr = INADDR_ANY;
bzero(&(server_addr.sin_zero),8);

if (bind(sock, (struct sockaddr *)&server_addr, sizeof(struct sockaddr)) == -1) {
    perror("Unable to bind");
    exit(1);
}
if (listen(sock, 5) == -1) {
    perror("Listen");
    exit(1);
}

printf("\nTCPServer Waiting for client on port 5000");
fflush(stdout);
strcpy(tInfo.messageID,"");
```

LOG FILE CHECK TO RESUME TRANSFER

```
if(access("logClient", F_OK )==0){
    fp = fopen("logClient", "rb");
    fread(&tInfo, sizeof(tInfo), 1, fp);
    fclose(fp);
}
```

SYNCHRONIZE WITH SERVER

```
if(strcmp(tInfo.messageID,argv[1])==0){
    printf("==== CHECKPOINT 6 ====\n");
    if(tInfo.cS.transferCompleted){ // File transferred completely
        printf("File totally transferred\n");
        send(sock,"closeSock",strlen("closeSock"), 0);
        close(sock);
        return 0;
    }else{ // File partially transferred
        // False transfer justto increase the counter
        send(sock,tInfo.messageID,strlen(tInfo.messageID),0);
        printf("Current partition = %d\n",tInfo.cS.packetNo);
        // Send offset
        char tempOffset[10];
        itoa(tInfo.cS.packetNo,tempOffset,10);
        printf("Sent offset = %s\n",tempOffset);
        send(sock,tempOffset,strlen(tempOffset),0); // Send buffer data to server
        bytes_recieved=recv(sock,recv_data,1024,0);  // Wait to receive response from server
        recv_data[bytes_recieved] = '\0';
        //~ printf("Response = %s\n",recv_data);
        if(strcmp(recv_data,"savedoffset")!=0){ // response appended because we didn't receive the response after false transfer
            printf("Server offset not received\n");
            send(sock,"closeSock",strlen("closeSock"), 0);
            close(sock);
            return 0;
        }
    }
}else{ // New file
```

NEW CONFIGURATION INITIALIZED

```
    printf("==== CHECKPOINT 7 ====\n");
    time(&tInfo.transferStartedAt);
    strcpy(tInfo.messageID,argv[1]);
    tInfo.cS.transferCompleted=0;
    tInfo.cS.packetNo=0;
    tInfo.noPackets=0;
    //~ printf("FileName = %s\n",tInfo.messageID);

    // Send server new configuration
    send(sock,tInfo.messageID,strlen(tInfo.messageID),0); // Send buffer data to server
    bytes_recieved=recv(sock,recv_data,1024,0);  // Wait to receive response from server
    recv_data[bytes_recieved] = '\0';
    if(strcmp(recv_data,"saved")!=0){
        printf("Server configuration not saved\n");
        send(sock,"closeSock",strlen("closeSock"), 0);
        close(sock);
        return 0;
    }
    // Send offset
    char tempOffset[10];
    itoa(tInfo.cS.packetNo,tempOffset,10);
    printf("Sent offset = %s\n",tempOffset);
    send(sock,tempOffset,strlen(tempOffset),0); // Send buffer data to server
    bytes_recieved=recv(sock,recv_data,1024,0);  // Wait to receive response from server
    recv_data[bytes_recieved] = '\0';
    if(strcmp(recv_data,"offset")!=0){
        printf("Server offset not received\n");
        send(sock,"closeSock",strlen("closeSock"), 0);
        close(sock);
        return 0;
    }
}
```

Client send required details to server for transferring a new file

PARTITIONS CALCULATED TO READ TOTAL FILE

```
file = fopen(tInfo.messageID, "rb"); // File opened to read
fseek(file, 0, SEEK_END);
long fsize = ftell(file);
fseek(file, 0, tInfo.cS.packetNo);
tInfo.fileSize=fsize;  // Filesize initialized
char temp[bufferSize];  // Buffer array initialized (Don't change in between transferof one file)
int partitions=fsize/sizeof(char);
//~ printf("Partitions = %d\n",partitions);

// ctime(&transferStartedAt);
```

File size checked and number of partitions calculated

SEGMENT BY SEGMENT DATA TRANSFER

i => file read

j => buffer

```
for(i=0,j=0; i < (partitions); i++,j++) { // File reading byte by byte and store in buffer for transmission
    if(j==bufferSize){
        send(sock,temp,bufferSize,0); // Send buffer data to server
        bytes_recieved=recv(sock,recv_data,1024,0);  // Wait to receive response from server
        recv_data[bytes_recieved] = '\0';
        if(strcmp(recv_data,"received")==0){
            tInfo.noSegments+=1;
            fp = fopen("logClient", "wb");
            fwrite(&tInfo, sizeof(tInfo), 1, fp);
            fclose(fp);
            // sleep(1);
        }
        printf("\nRecieved segment = %d\n" , tInfo.cS.segmentNo);
        // fflush(stdout);
        j=0; // Reset buffer

    }
    fread(&temp[j],sizeof(char), 1, file); // Overwrite/New write to buffer
    tInfo.cS.segmentNo=i;
    if((i+temoSegmentNo)==fsize){
        break;
    }
}
```

SEGMENT NOT FILLED TOTALLY

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

3 segment =
3*10 = 30 bytes

REST PART OF SEGMENT

```
if(j!=0){ // Rest part of buffer data transmitted if datasize is less than buffersize
    send(sock,temp,j, 0);
    bytes_recieved=recv(sock,recv_data,1024,0);
    recv_data[bytes_recieved] = '\0';
    if(strcmp(recv_data,"received")==0){
        tInfo.noPackets+=1;
        fp = fopen("logClient", "wb");
        fwrite(&tInfo, sizeof(tInfo), 1, fp);
        fclose(fp);
        printf("Sent packet = %d (Rest part)\n",tInfo.noPackets);
        sleep(1);
    }
    // printf("\nRecieved data = %s\n" , recv_data);
    // fflush(stdout);
}
```

CLOSE SOCKET ON REQUEST FROM CLIENT

```
if (strcmp(recv_data , "closeSock") == 0 || strcmp(recv_data , "transferCompletedcloseSock") == 0){
    printf("==== CHECKPOINT 1 ====\n");
    //~ fp = fopen("logServer", "wb");
    //~ fwrite(&tInfo, sizeof(tInfo), 1, fp);
    //~ fclose(fp);
    //~ transferSequence=0;
    close(sock);
    return 0;
    //~ break;
}
```

NEW CONFIGURATION FROM CLINET

```
if(transferSequence==1){
    printf("==== CHECKPOINT 3 ====\n");
    if(strcmp(tInfo.messageID,recv_data)!=0){
        strcpy(tInfo.messageID,recv_data);
        time(&tInfo.transferStartedAt);
        tInfo.cS.transferCompleted=0;
        tInfo.cS.packetNo=0;
        //~ fp = fopen(tInfo.messageID, "wb");fclose(fp);
        send(connected, "saved",strlen("saved"), 0);
        transferSequence=2;
        //~ printf("FileName = %s\n",tInfo.messageID);
    }
```

Configuration received from client

TRANSFER COMPLETED CONDITION AT SERVER SIDE

```
if (strcmp(recv_data , "transferCompleted") == 0){
    printf("==== CHECKPOINT 2 ====\n");
    time(&tInfo.transferCompletedAt);
    tInfo.cS.timeElapsed=difftime(tInfo.transferCompletedAt, tInfo.transferStartedAt);
    tInfo.noPackets=tInfo.cS.packetNo;
    tInfo.cS.transferCompleted=1;
    //~ printf("FileName = %s\n",tInfo.messageID);
    file = fopen(tInfo.messageID, "rb"); // File opened to read
    fseek(file, 0, SEEK_END);
    long fsize = ftell(file);
    fseek(file, 0, SEEK_SET);
    tInfo.fileSize=fsize;  // Filesize initialized
    printf("==== CHECKPOINT 5 ====\n");
}
```

Update the complete status of transfer in log

DATA WRITING TO FILE AT SERVER SIDE

```
}else if(transferSequence==2){
    printf("==== CHECKPOINT 6 ====\n");
    printf("Offset string = %s\n",recv_data);
    tInfo.cS.packetNo=atoi(recv_data);
    printf("Offset integer = %s\n",recv_data);
    send(connected, "offset",strlen("offset"), 0);
    if(tInfo.cS.packetNo==0){
        printf("File reset\n");
        fp = fopen(tInfo.messageID, "wb");fclose(fp);
    }
    transferSequence=0;
    readOffset=tInfo.cS.packetNo;
}else{
    printf("%s",recv_data);
    fp = fopen(tInfo.messageID, "ab");
    for(k = 0; k < bytes_recieved; k++) {
        //~ printf("%c",recv_data[k]);
        fwrite(&recv_data[k],1,sizeof(char),fp);
    }
    fclose(fp);
    tInfo.cS.packetNo+=1;
    //~
    // fflush(stdout);
    send(connected, "received",strlen("received"), 0);
}
}
```

For partial transfer in this section offset is received from client

Data written to file along with the transmission

THANK YOU

Is there any query?