Report

# Introduction

Application protocol design is a computer networking aspect that deals with layering the communication format between the server and clients connected to the server. Designing the protocol involves specifying how the data is structured, transmitted, and interpreted by the receiving socket.

In this report, we will discuss several key considerations we implemented in designing our protocol. Firstly had to design a way that our client was going to communicate with the server through socket programming. This was implemented by defining message protocols in the application layer.

# Server-Client connection

There are two types of connection-oriented services or transport layer protocols, it is the TCP and the UDP connection. The TCP is said to be a reliable data transfer service, where the sent message through the socket is guaranteed to be received without error and in proper order. The UDP, on the other hand, is unreliable and data sent may not be received and if received, it may arrive out of order.

Since our task was to create an application that sends and downloads files to/from the server. It was paramount for us to use the TCP service since it is reliable and all bytes of the file must be in order. We used this line of code to establish a TCP connection through the socket → client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) for the client and → server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) for the server.

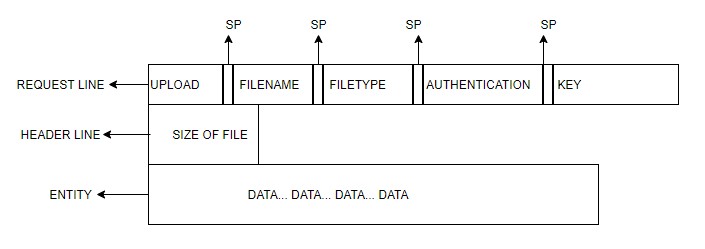
Even though the TCP service is reliable, we had to design a file validation mechanism. For this, the sender first sends the file size of the file and the receiver uses the file size as a condition in a while loop to receive the bytes. This is done in this Format → while int(len(data))<int(file\_size): mess = conn.recv(4096) data +=mess

# Framework of communication

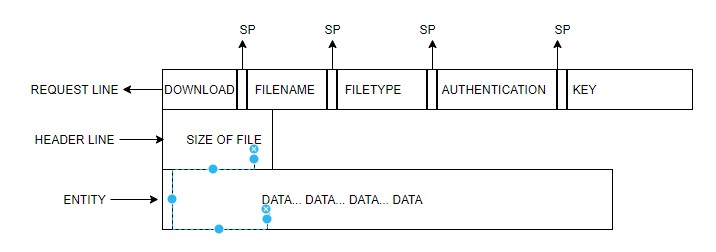
After the connection has been established, we designed specific requirements and constraints that ensure privacy/authentication and reliability in terms of verifying that messages are delivered correctly.

For this, we will discuss three main structures of our messages that were implemented in the protocol. Namely, command messages, data transfer messages and control messages.

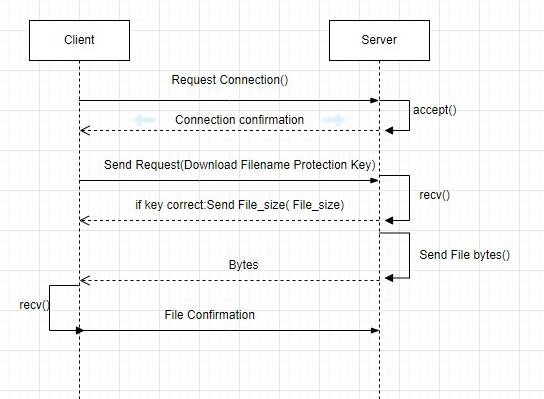
1. **Commands** → the commands used are mainly the request line which is in the format METHOD(download/upload) | filename | FILETYPE(image/video/textfile) | AUTHENTICATION(protected/open) | KEY(10digits). This request line is sent by the client to the server and is part of the initiation of communication after the connection has been successful. We have implemented a list files command which the client sends to the server and the server lists all downloadable files.
2. **Data transfer** → messages of this type are used to carry out data that is exchanged between both parties. In our design, data transfer messages include the .send(1024).decode(‘utf-8’), and .recv(1024).decode() messages.
3. **Control messages** → manage dialog between parties by being acknowledgments and retransmission requests. In our protocol, there's an acknowledgment of every message sent and error messages which form part of retransmission requests. Examples of such fileNotfound error, file received an acknowledgement, sizeoffile received, etc.



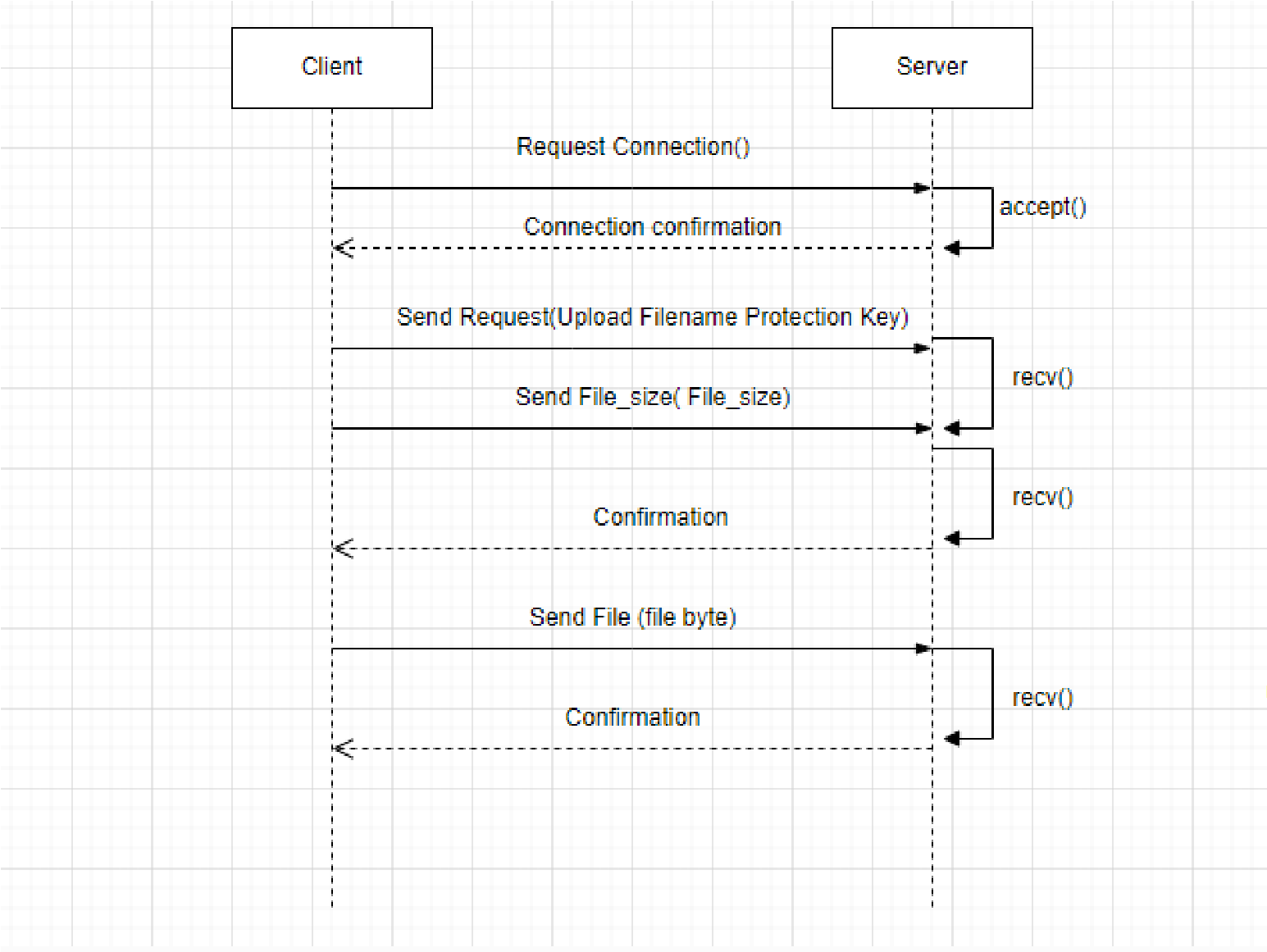
**Figure 1.1 General format of our Protocol request message for uploading.**



**Figure 1.2 download request message with the size of the file and data response by the server.**



**Figure 1.3 represents the flow diagram of communication when there's a download request.**



**Figure 1.4 represents the flow diagram of communication when there's an uploaded request**

# Server Features

**Method features:**

1. **The upload feature**: in this feature the client is given a right to post files to the server and specify if they want these files to be open for everyone or if they would like to keep them private and only give access to users with the key to the files. This feature enables users to use the server as their cloud storage or to send files to other users.
2. **The download feature**: the users can download files that are stored on the server. These files can either have keys to access them given that they are protected and some require no key if they are open for everyone to access. To get the protected files users should provide the right key for the file.
3. **The search feature**: users connected to the server can search for files that are available for download to anyone
4. **Storage feature**: We have implemented storage for files to be folders, one folder for open files (files that can be accessed by any client), and protected files (these are files that require a key to access them).

**Client Report**

The client I designed creates a TCP socket, connecting with the server. The user types in the request line in a command prompt in the format:

→Download: Download Filename Filetype Protection\_Type Key

→Upload: Upload Filepath Filetype Protection\_Type Key

This file request line is sent to the server, if it is a download request, the server sends the file size which the client will use to receive the number of bytes in a file. If it is an upload request, the size of the file must be sent to the server before the file is sent and When uploading the file path must be specified.

Every sent message is encoded with the format ‘utf-8’ and every received message is decoded with the same format. This allows strings to be converted to bytes so that they can be sent through the socket.