A STEP-Based File Uploading Application

Chenkai Zhou

2033555
Information and computing science
Xi'an Jiaotong Liverpool University
Suzhou, China
Chenkai.ZHOU20@student.xjtlu.edu.cn

Haoyang Lin

2036741
Information and computing science
Xi'an Jiaotong Liverpool University
Suzhou, China
Haoyang.Lin20@student.xjtlu.edu.cn

Jiayue Wang

2034912
Information and computing science
Xi'an Jiaotong Liverpool University
Suzhou, China
Jiayue.Wang20@student.xjtlu.edu.cn

***Abstract*—Network-based file transmission plays an indispensable role in contemporary society. The Simple Transfer and Exchange Protocol(STEP) is a simple TCP-based application-layer protocol, which stipulates the data and file transfer in the network. The project aims to design a client application to implement the file transfer between ready-made server end and client end by STEP. By fixing the bugs of the server and implementing the functions including authorization, fetching token, and uploading files, we succeed in designing a preliminary version client application. The project provides a simple network-based file transfer demo for future network research and professional optimization.**

Keywords—computer network, file transfer, transfer protocol

I. INTRODUCTION

With the rapid development of network technology, an increasing number of applications need to exchange files and data over the network, based on the FTP or HTTP protocol, which are on the layer of TCP[1]-[3]. They are widely used because of the high reliability of TCP in data transmission. The project proposes a new protocol called Simple Transfer and Exchange Protocol(STEP) to implement the function of file transmission. The STEP protocol is an application layer protocol, which is based on TCP protocol as well. STEP stipulates its own format of request/response messages on functions including user login, file/data uploading, file/data downloading, and file/data deleting.

The project focuses on the implementation of the uploading file function based on STEP. A client application is designed by us to upload files to a given server application depending on STEP. A typical server-side and client-side model is used to complete a system with file transfer functionality[1]-[3]. In this system, the server can receive files from the client[4]. Python language is used to finish our project. We need to fix all the bugs in the Python file of the given server application and run it as our server. After finding all the bugs, a connection between the client side and the server side will be established. Then, the client tries to log in to the server system and finish its upload tasks by the authorization token and upload plan defined by the server.

Initially, we encountered several challenges： First, the server can not run without correctly debugging. Second, there is no connection between the server and the client application. Third, we should use the new STEP protocol with special format request messages to get authorization and upload files to the server.

Moreover, we find several relevant practices concerning the project:

1. The simple application of file transfer on different virtual machines
2. A simple demo with core concepts of network-based transferring files for future development.

Finally, we succeeded in accomplishing the following three points:

1. Find all the bugs and debug them successfully.
2. The connection between the server and client is successfully established.
3. The files can be uploaded according to the token.

.

II. RELATED WORK

The number of related protocols and practical applications for file transfer is extremely large, so we briefly review some classical and common instances, which are instructive to our application design on STEP.

File transfer protocol (FTP) is a representative standard protocol related to file transfer[1]-[2]. Most commercial file transfer clients and servers are implemented according to this standard specification for compatibility. There have been multiple types of research in this field. Naturally, academia has also developed a considerable interest in it.

One related application is Xftp, an FTP, SFTP file transfer program for the Windows platform that helps users to securely perform fast file transfer tasks on Unix and Windows systems[5]. It allows visualization of the file list, which is more in line with Windows users' usage habits.

All the operations we perform on Xftp can be done using Xshell[5], except that for the same purpose, Xftp can be achieved through a visual interface and by clicking and dragging, whereas Xshell requires the use of commands by hitting them.

In the oil and gas industry, data management is a huge challenge, utilizing TCP parallel data connections and scheduled transfers as advanced user options when transferring data[6].

III. DESIGN OF THE SYSTEM

The whole system consists of two core modules: client application and server application. Our design focus on the design of the client part.

A. C/S Architecture

Figure 1 shows the basic Client-Server(C/S) architecture. The client end and the server end are on 2 different virtual machines with different IP addresses to simulate the real application scenario. The client application sends STEP messages to the server to request to log in or upload the file,

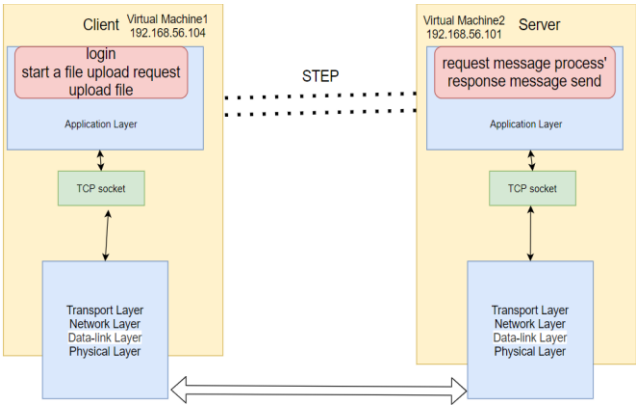

Fig. 1. C/S architecture diagram.

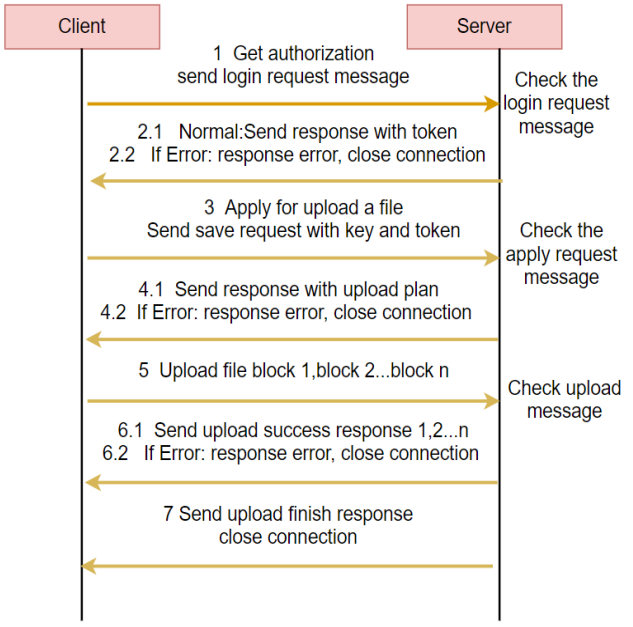

Fig. 2. System workflow.

and the server sends back STEP response messages according to different types of requests.

B. System Workflow

Figure 2 illustrates the workflow of the whole system.

Firstly, the client application needs to get authorization and login to the server, so it sends a login(AUTH) request to the server. After receiving the login(AUTH) request, the server will check the fields of the message. If the login request is valid and in the correct format, the server will respond with a token, otherwise, it will send an error response with related error descriptions and close the TCP connection. The token plays a role like authentication information, which means the client is valid and authorized. The token should be included as a field of all following request messages to prove authorization.

Secondly, the client gets authorization after receiving the token, and it will apply for an upload file request by sending a SAVE request with the received token, file size, and a key (file name).After receiving the upload application, the server will check the format and content of the SAVE request

Algorithm 1: Get Authorization

Input: *server_ip, port, id*

Output: *token,client_socket*

```
client_socket = createTCPSocket()
client_socket .connected (server_ip, port)
requestMessage = generateLoginRequest(id)
client_socket .sendToServer(requestMessage)
token = serverResponse(requestMessage)
If token is correct
    return token, client_socket.
```

Algorithm 2: Apply Upload and Upload File

Input: *token, client_socket, id, file_path*

Output: upload apply and upload file, no output

```
uploadApply = applyMessage(id,token, file_path)
client_socket . sendToServer (uploadApply)
uploadPlan = serverResponse(uploadApply)
If uploadPlan is Not None
    name, size, total_block,block_size= uploadPlan
    for block in total_block do
        If block is the last block
            client_socket. sendToServer(size)
        else
            client_socket. sendToServer(block)
            size = size - block
```

message. If the format or fields information is wrong, the server will send related error responses to the client and close the connection. Normally, the request is correct, and the server will calculate the upload plan according to the file size and pre-allocated block size and send it as a response.

Thirdly, after receiving the upload plan, the client starts to upload the file. The client uploads the file through UPLOAD request according to the just received upload plan. The file will be split into serval parts according to the upload block size which is defined in the upload plan. With the help of TCP, the client sends them in sequence, and the server will receive the file block in sequence. The client will send only one block in every UPLOAD request, and the server will responses every request. If the whole file is received by the server, which means the uploading process is done, the server will respond finish information with md5 checking and finally close the connection. If any error occurs in the process of uploading, the server will send an error message and close the connection as before

Algorithm 1 demonstrates the process of the client application logging into the server and fetching the token. The client first creates the TCP connection with the server

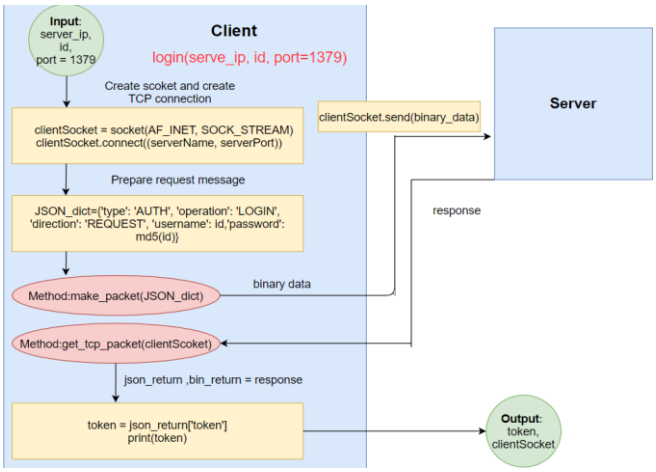

Fig. 3. Login and fetch token

and send the login request. Algorithm 2 shows the applying upload and uploading file process. After logging in and fetching the token, the client application applies for upload a file with a specific file path and token. The upload plan with block size and block number from server will instruct the client to upload the file block by block. The client send block size bytes every time, and it will send all the rest bytes(the last time) if the rest bytes can be contained in one block

IV. Implementation

In this section, we report the detailed process of our algorithm and actual implementation of system in Section 3.

Setup

The developed host is equipped with Intel(R) Core(TM) i5-1035G1 CPU and 16 GB RAM. For the sake of convenience, we develop our application on Windows 10 OS and use the remote connection (SSH) to test our code on virtual machines(Ubuntu OS). Our virtual machines are equipped with 4GB RAM.

All the source codes are implemented in Python 3.10 and developed on JetBrains PyCharm. These Python libraries below are used in our implementation(client application) process: *hashlib, json, os, struct, argparse, socket, time*

A. Implementation Steps

Figure 3 is the program flow chart of login and fetch token, and it demonstrates the detailed implementation of the get authorization function. The function of getting authorization is implemented by a method called *login*, this method should receive 2 or 3 parameters: the IP address of the server, student ID number, and server port number(default 1379). The program will create a client TCP socket with the parameters and connect with the server. After the connection is established, the program will prepare the STEP request message in JSON format. According to the STEP, The keys 'type', 'operation', and 'direction' should be 'AUTH', 'LOGIN', and 'REQUEST', and the keys ‘username’ and ‘password’ should use the student id and the haslib library generated md5 value of id. In the next step, the program will call *make_packet* method to convert the python dictionary object to bytes, and send to the server. After the server responds to the client, the program will call *get_tcp_packet* method to convert the byte

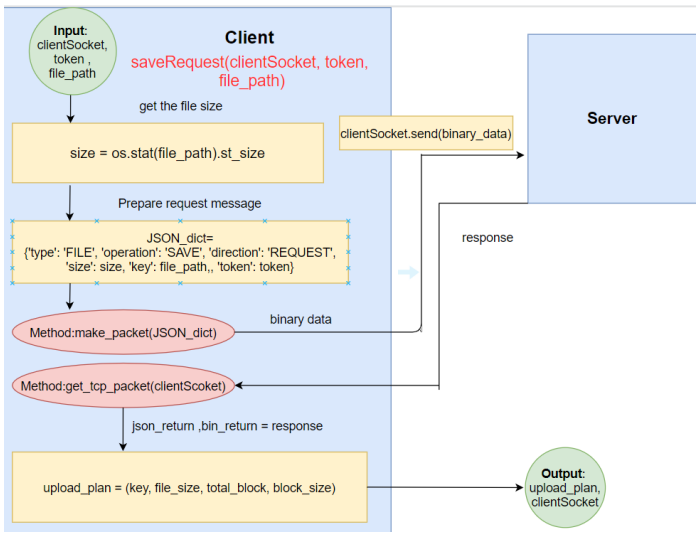

Fig. 4. Apply for uploading a file

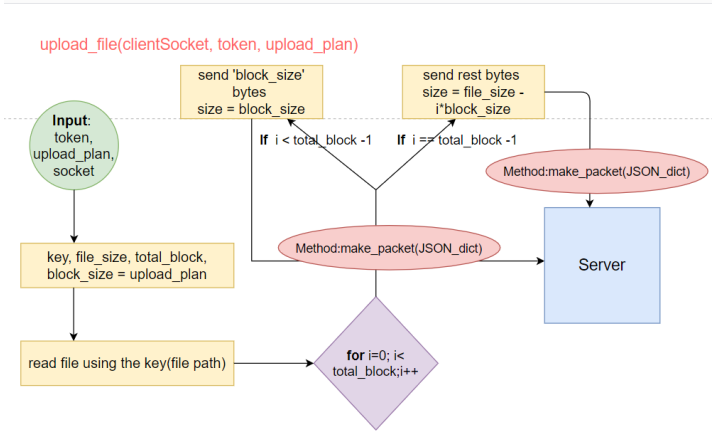

Fig. 5. Upload a file

response message to Python tuple and get the token. Finally, it outputs the client socket and token for other methods to use.

Figure 4 is the program flow chart of uploading application, and it demonstrates the detailed implementation of the applying for upload function. The function is implemented by a method called *saveRequest.* This method take file path, and output token, and socket from the *login* method as parameters. The program first calculates the file size through Python built-in library *os,* and prepares the STEP request JSON dictionary as before. The key ‘type’, and ‘operation should change to ‘FILE’ and ‘SAVE’. The key ‘size’, ‘key’, and ‘token’ should be pre-calculated file size, file path, and the output token from the previous method. Afterwards, the program will call *make_packet* method and send the binary data to the server as before mentioned. After receiving the response message from the server, which means that the server allows this upload application and generates a related upload plan, the program call *get_tcp_packet* method and obtains e the upload plan from the *json_return.* The upload plan includes 4 critical fields: file path, file size, the total number of blocks, and the size of every block. At last, the method will output the client socket and a tuple of these 4 fields.

Figure 5 describes the actual implementation of the function of uploading file to the server. The function is implemented by a method called *upload_file.* The method

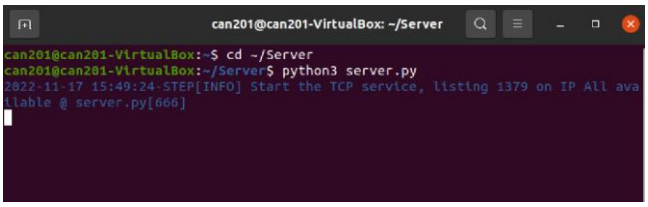

Fig. 6. Snapshot of sever debugging

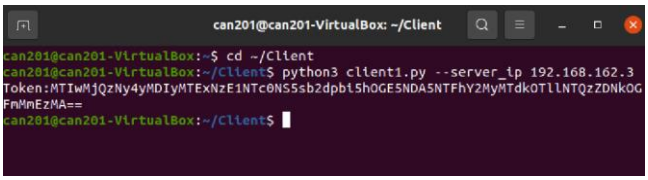

Fig. 7. Snapshot of getting authorization

Fig. 8. Snapshot of successfully uploading a file

receives the token, the upload plan from the previous step, and the client socket as the parameters. The program first analyzes the upload plan to parse it into 4 variables, which should be used afterward. Then the program read the file with the key(file path) into a byte object. Afterward, the program starts a for loop to upload the file to the server. The program prepares the JSON request message with the upload size and calls the *make_packet* method to packet the byte data. In every loop, the client should send a block to the server, so the size field is the block size. If it comes to the last loop, the size of the rest file which not be uploaded is smaller than one block size, it will send all the rest bytes in one block and the size field in JSON message will be changed to be the rest bytes size. During the uploading process, the server will respond to the client after receiving every upload request to inform the status. When the uploading process finish, the server will respond to the client with an md5 check and close the TCP connection.

B. Programming skills and difficulties

We use Object-Oriented Programming to implement our client application. We encapsulate a class *Client*, which has three methods as its member methods. The *Client* instance object is created every time when the user wants to upload a file to the server. At the server end, the pre-made code uses parallel skills by starting threads to connect with clients. The most difficult thing is the use of unfamiliar Python libraries. However, it is solved by teamwork and looking up official documents.

V. TESTING AND RESULT

The testing environment is the same as the implementation environment. Especially, the Operating System is Ubuntu(64-bit), the RAM is 4096MB, and the Python version is 3.10.

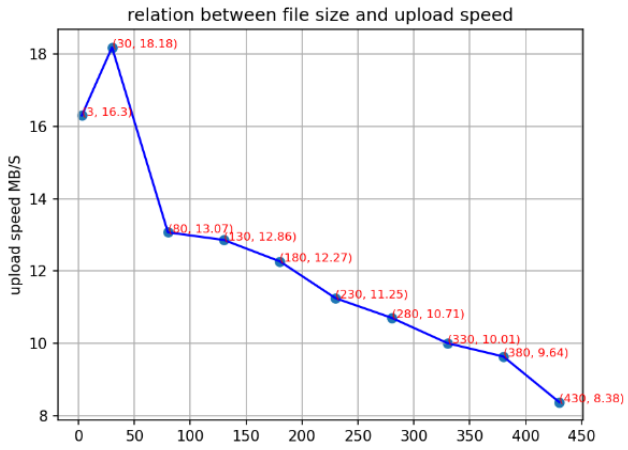

Fig. 9. Relation between file size and upload speed

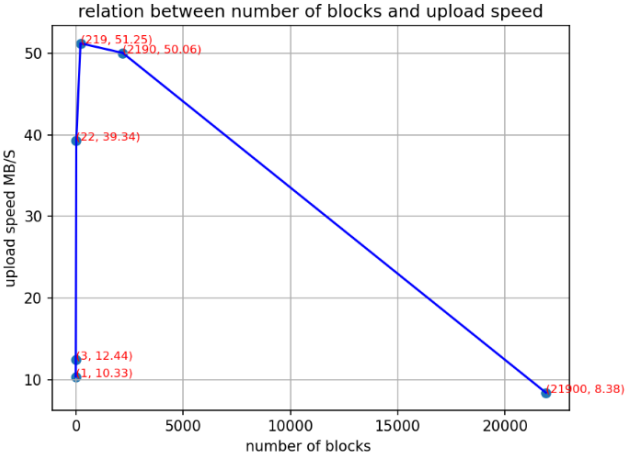

Fig. 10. Relation between the number of blocks and upload speed

Firstly, Several bugs in the server have been fixed, which are: adding “import hashlib” at the top of the file, replacing “main” with “main()”, adding “th. start” in the *tcp_listener* method, replacing “data_process” with “file_process” in the *step_service* method, replacing “SOCK_DGRAM” with “SOCK_STREAM”, replacing ‘410’ with ‘405’ and replacing “STEP_service” with “step_service” in the thread module. Figure 6 shows the result after successful debugging. Secondly, through python programming, the server-side authorization is successfully obtained and a token is returned with the input of IP address, port number, and student id number. The result is illustrated in Figure 7. Thirdly, Figure 8 demonstrates that after getting the authorization and applying for the uploading, the uploading plan is returned. Then file uploading has been implemented, and return the required information, including file name, block index, transfer information, MD5, etc.

In addition, we conducted several experiments to further test the performance of the client application. In order to assess the overall file upload speed, we measured the amount of time needed to upload files of various sizes, ranging in size from 3 MB to 430 MB. To lessen the unintentional error brought on by network fluctuation and other factors, an average of 20 measurements was employed to calculate the upload speed. The results are displayed in the following

Figure 9, which is drawn through *matplotlib*. It is found that the upload speed is high while the file is small, with a slump between 30 MB and 80 MB, then it slows down as the file size increases. In another experiment, we compared how varying block sizes affected the speed of transferring the same file. We modified the block size from 20480 to 2048000000 and chose a 430 MB compressed file as the transmission file. For clarity, the number of packages are used for evaluation, and the larger the block size, the smaller the number of blocks. In each case, we measured 20 times in the same network environment and averaged the results. It was discovered that when the block size was moderate, the transmission speed was rapid as opposed to when the size was excessively large or tiny. Figure 10 shows the result. The client and server in the aforementioned experimental setup were running on separate virtual machines on the same computer. Additionally, we did an experiment where the client and server uploaded files while running in separate virtual machines on two different computers. We discovered that the transmission speed was essentially the same as that of the earlier experiments. Finally, we tested the upload speed of different types of files of the same size. For instance, the file size is 430MB, the default block size is 20480, the file types include.zip,.jpg,.txt,.pdf, etc., and found that the transmission rate is basically the same

VI. CONCLUSION

In this project, we developed a client application based on the given STEP protocol and implemented the function of uploading files to the server through Python socket programming. The steps involved in implementation are as follows: first, the server-side bugs are addressed; next, we successfully request authorization from the server side; and finally, the file uploading function using socket programming is implemented.

However, the project still has the following shortcomings that can be improved in the future: 1. It only has a file-uploading function, further functions which are defined in STEP like downloading and deleting could be implemented. 2. In our project, only one file can be transferred at a time, how to transfer a large number of files at the same time rather than pack them as a zip file should be considered. 3. The program can only run with the command line input, which is complex for users who have little Python knowledge, so a straightforward and simple user interface should be designed

ACKNOWLEDGMENT

Chenkai Zhou(2033555), Haoyang Lin （ 2036741 ） Jiayue Wang（2034912） contribute same to the project

REFERENCES

[1] J. Postel and J. K. Reynolds, File Transfer Protocol, 1980, [online] Available: https://tools.ietf.org/html/rfc765..

[2] P. Hethmon, Extensions to FTP, 2007, [online] Available: https://tools.ietf.org/html/rfc3659.

[3] R. Khare, "The transfer protocols," in IEEE Internet Computing, vol. 2, no. 2, pp. 80-82, March-April 1998,.

[4] Jingfeng Zhang, "Research on image transmission system based on 3G communication platform", Sensors and Transducers, vol. 174, no. 7, pp. 187-192, January 2014.

[5] Yulong You “Upload tool on Windows platform”, vol.2, no.2, pp.97-98 March 2003

[6] F. de Souza, M. Salles, F. Campos, S. Costa, M. Costa, and N. Ebecken, "Specialized File Transfer Service for Large Oil and Gas Datasets," 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, 2013, pp. 176-177.