File Upload Application Based on Protocol STEP

Yizhen Cao, Yuzhen Chen and Bowen Li
School of Advanced Technology
Xi'an Jiaotong-Liverpool University
Suzhou, P.R.China
{Yizhen.Cao20,Yuzhen.Chen20,Bowen.Li20}@student.xjtlu.edu.cn

***Abstract*—People become beneficiaries owing to file transmission function built in certain software. File transfer technology helps conveniently store and retrieve data, protect environment by reducing the paper use, etc. In this project, we are motivated to develop a file-uploading application with a newly-defined application-layer protocol by using Python. Thus, we design the overall structure of the application and core algorithm related to authorization and file upload. Then, we do the practical Python programming on a specific physical host to implement the algorithm designed in the previous stage. Ultimately, we test whether this application can realize upload function normally and the performance of file upload on the testbed. The testing results show that our application can work and the upload performance is efficient.**

***Index Terms*—file upload, authorization, Python, STEP**

I. Introduction

Distinct types of versatile file-uploading WEB applications have penetrated into all walks of life since the emergence of Internet, which fulfills access to the files stored on the server at any time for both private use and industrial demands [1], and it highlights the practical significance of file transfer in computer networking area.

In this project, we are provided a server-side application based on a simple predefined protocol named STEP. Then we aim to develop a client-side application using Python that implements file upload function with C/S networking architecture based on the same protocol. There are two major challenges to be coped with. First, the team members are unfamiliar with STEP since it is a newly-defined protocol. Second, developers are unskilled with part of Python modules needed to be used. Due to the limited storage of mobile phones and personal computers, a substantial amount of people preserve files using cloud storage and file uploading is an indispensable process of the technology. Thus, this project can potentially be applied in cloud storage and all the practical applications requiring file uploading.

The contributions of our work are listed in the following:

- we fixed all the bugs occurred in the existing server-side code which completed all the services provided by STEP.
- we accomplished a code module to get token from the server side for authorization.
- Third, we fulfilled the intact function of file upload based on STEP in the client-side application.

The rest of the report are organized as follows. Section II presents some related work. Section III illustrates the general architecture of file-uploading application and fundamental algorithm. Section IV elaborates actual implementation environment and presents part of core source code. Section V demonstrates the testing procedure and corresponding result plus the figures showing upload.

II. Related Work

In this section, the related work about file processing and transmission problems across the Internet is reviewed from function realization to performance test.

A. Implementation of function

The file transmission and processing functions are one of the main functionalities realized through different application layer protocol. From the perspective of file transmission reliability, several work has been proposed. For example, Yubing.Y et al. [2] add file sending and receiving agent in order to encode, decode, format convert, and rename the files which ensures the correct order and dependability of reverse transmission through file time sequencing and window control. On the other hand, other work pays more attention on improving the efficiency of C/S structure file transmission through multi-threading. For instance, Hyoungyill Park et al. [3] mentioned that the high-efficiency network can be achieved to enable the high speed of HD video file using the connection with networks between hosts through packet creation and multi-session of Parallel TCP in order to maximize the effectiveness of such a network.

B. Testing of performance

Performance test is also an important region in file transmission research which cannot be ignored. At functional testing level, several works can be done to verify the completeness and accuracy of the function. For instance, Yao Hu [4] proposed that the functional completeness can be tested by comparing the effect of file transfer in different sizes and formats.As to performance testing, Chang [5] mentioned that the amount of data flowing through per unit time can be monitored to assess transmission efficiency.

The aforementioned related work provided the theoretical basis for the implementation of file upload functionality of this project, and helped us to clarify the methodology for testing program performance.

III. Design

Client-Server architecture is one of the high-efficient network architectures that perform both the functions of client and server so as to promote the interaction of information between them [6]. It enables several users to access the same database simultaneously, and the database will store massive amount of data from client side. The design concepts of this project is highly dependent on the request-response interaction between client and server which is the central challenge of algorithm design.

A. C/S network architecture

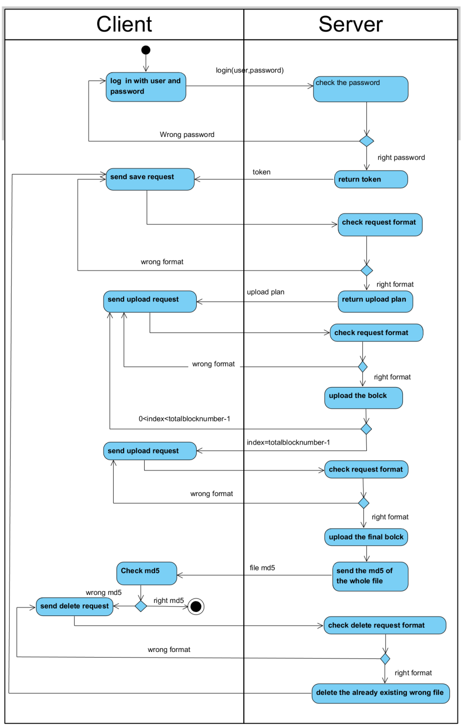


Fig. 1. Activity diagram of C-S architecture

According to Fig1, we can abstract the components of this simple network into two classes: client and server. Before a client uploads a file to the server, it requires to send a request packet which contains user name and password to the server to obtain upload authorization. Once the client has received the authorization token, it will send a save request message with the received token to server in order to get the upload plan. After that, the client will divide the uploading file into blocks according to the upload plan and send them block by block. Finally, the server will return the md5 value of the uploaded file to the client to check the completeness and correctness of the file.

B. The workflow of solution

Since the functionality mainly focuses on authorization and uploading, we can do some requirement engineering from these two aspects. Firstly, we will define the specific implementation flow of each function and then abstract out the specific functions each functionality needed.

1) Authorization and fetching token: During authorization, the client should firstly send login request to the server with username and password. According to the STEP protocol, the password corresponding to each user name is the md5 value calculated from the user name, therefore, A function get_username_md5(username) that calculates the md5 value is essential for the implementation. Since every request packet sent by client share the same format, make_packet and make_response_packet are also needed. Once the server received the request, if there help client correct mistake. Otherwise, the right token will be returned from server which means login successfully. Function login_get_token is designed to implement fetch token and exception handing.

2) Uploading the file: After the client obtains the token successfully, the client will send save request to the server using key, token and file size. The key we defined here is just file name since its uniqueness and ease of differentiation. After that, the server will check the format of request_packet and send response_packet. If there are some errors, the server will ask the client to resend the request. Otherwise, the server will return the upload_plan which is used to instruct client to upload the file. Due to the size limitation of each send, file block technology will be used while transmitting big file. Therefore, the upload plan should contain file size, total number of block and the size of each block. Function get_upload_plan will be designed to realize the above functionality.

Once the client receives the upload plan, the client will send the first block to the server, similarly, the server will also check the format of request_packet and send response_packet. If there exists some errors, the server will ask the client to resend the request. Otherwise, the server will return the new upload_plan which is used to instruct client to upload the next block. These steps will be repeated until the client finishes uploading the last block.

While the server side has downloaded the whole file, it will send the md5 value back to the client which aims to check the completeness and correctness of the file. If the client receives wrong file_md5_value, the client will delete the existing file in server side and then resend the save request to upload the file again. Otherwise, the complete file has been uploaded successfully. According to the above discussion file_delete and file_upload should be included in Algorithm Design.

C. Algorithm Design

Here are some pseudo codes which indicate the detailed information of each function.

1) Authorization:

```
login_get_token(username,connection_socket):
 get password according to username
 send login request to with user name and password
 while true:
     get json_data and bin_data from server side
     if status code of response_packet is 200:
         fetch token
         jump out of the loop
     else:
         resend login request to server
         continue to next-time loop
```

2) File uploading:

```
file_upload(username, connection_socket, file_path, token):
get upload plan
if upload_plan is empty or none:
     close connection
open the file according to the file path and read it
divide the file into k block
for i in range (0, k):
     send the block to the server
     while true:
         if the block is the last block of the file
             check MD5 value of the whole file
             if there are some errors in file_md5
                 delete and reupload the file
         if there is something wrong with the token:
             resend the upload request to the request
                 if json_data[FIELD_OPERATION] == OP_UPLOAD
and json_data[FIELD_TYPE] == TYPE_FILE:
             if response_status is 410:
                 resend the upload_request
             if response_status is 200:
                 break
             else:
                 connection close
connection close
```

```
file_delete(connection_socket, token, key, username):
send file delete request to server
while true:
    get json_data and bin_data from server side
    If there's something wrong with token:
        resend the packet with delete operation
    else:
        if successfully delete or key does not exist:
            return nothing
        if something wrong:
            resend the packet with delete operation
```

```
get_upload_plan(connection_socket, file_path, token,
username):
if file_path is empty, then no file needs to upload
key is set to be the same as file name
send save request to server
while true:
    get json_data and bin_data from server side
    if key exists:
        return nothing
    if the status is 200:
        return upload plan
    if the status is 410:
        resend save request to server
        continue to next-time loop
```

IV. Implementation

In this section, we use Python to program all the components in the file-uploading application on the basis of the existing design section.

A. Implementation Environment

The physical host that we do actual code work is HUAWEI MateBook D 14 having CPU 6-core AMD Ryzen 5 45000U with Radeon Graphics with speed 2.3GHz and 16 GB main memory with speed 2667 MHz. The operating system installed is Windows 10 Home. Based on the host, the development tool Pycharm Professional Edition 2021.3.3, a prevalent Python IDE, is installed and used to complete the project tasks. Python modules used to implement this application are, *socket, json, os, hashlib, argparse, struct, time*.

B. Actual Implementation

We complete the overall application with the practical method of function by function, which means that one function achieves only one functionality needed in the file-uploading application. For example, Python function get_upload_plan() in the figure 2 only implements one thing, as the function name suggests, fetching file-uploading plan for the client. Figure 2 is the program flow chart demonstrating primary implementation steps and the Python functions are developed following the order as the figure shows. Socket is employed to fulfill networking communication based on TCP, the goal achieved in this application is file uploading and each component is divided into function in Python programming. Therefore, Socket programming, file operation and functional programming are major skills exploited in the project.

The two vital parts in this project are authorization and file uploading, each is implemented in a Python function, respectively. The Python code (only core part of source code) implementing authorization function is shown as below. The while loop aims to receive response packet after sending the login request to the server, after which the server should send response packet with a token to the user if the client password is correct and nothing is wrong. Thus, in the following code, we just fetch the token if the status code sent back equals to 200 and then break out the loop. Resend the request packet if

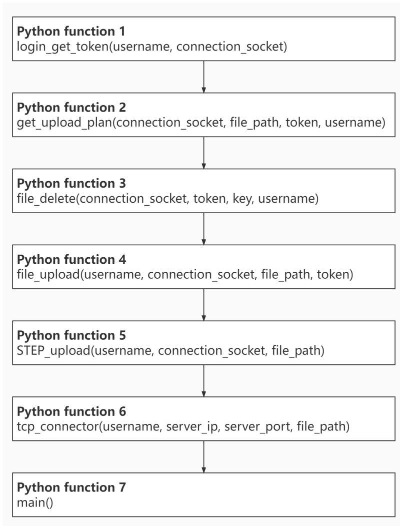


Fig. 2. steps of implementation

the response packet is not consistent as expected and jump to the next loop.

```
while True:
  json_data, bin_data = get_tcp_packet(connection_socket)
    if json_data[FIELD_OPERATION] == OP_LOGIN
and json_data[FIELD_TYPE] == TYPE_AUTH
and json_data[FIELD_STATUS] == 200:
        token = json_data[FIELD_TOKEN]
        break
    else:
        connection_socket.send(make_request_packet
(OP_LOGIN, TYPE_AUTH,FIELD_USERNAME:username,
FIELD_PASSWORD: password)
        continue
```

The code (only core part of source code) achieving file upload function is shown as follows. The variable is_last_block is a Boolean value set to identify whether the block sent is the last one. Open the file to be uploaded in read and binary mode since byte stream is used to transmit data. STEP protocol defines file upload must be block by block. Thus in each loop, f.seek(i*block_size) sets the pointer in the file to point to the start position of the block to be sent. Then, read one block file data if the block sent is not the last one, otherwise read the remaining data and set variable is_last_block to be True. Upload one block data in each loop, waiting for packet replied from server. At the beginning of the while loop, verify whether it is the last block sent because only the last successful response packet contains md5 value. Obtain md5 value if the status code is 200. Compare the md5 value received and calculated based on the local uploaded file. Send the delete request to remove the erroneous file already stored on the server side if the two md5 values are not identical and then recursively call function file_upload() to reupload. Close the client socket to terminate the upload application if nothing wrong happens. The source code that handles situation of block sent that is not the last one and receiving exception message is not demonstrated in the report since it is straightforward to implement. The corresponding algorithms have been listed in section III.

```
is_last_block = False
with open(file_path, 'rb') as f:
    for i in range(0, total_block):
    if.seek(i * block_size)
      if block_size * (i + 1) ¡ file_size:
        bin_data_send = f.read(block_size)
      else:
        bin_data_send = f.read(file_size - block_size * i)
        is_last_block = True
          plan.update(FIELD_BLOCK_INDEX: i)
          connection_socket.send(make_request_packet
(OP_UPLOAD, TYPE_FILE, plan, bin_data_send))
          while True:
          if is_last_block:
            json_data, bin_data_recv =
get_tcp_packet(connection_socket)
            file_md5 = get_file_md5(file_path)
                    if json_data[FIELD_OPERATION] ==
OP_UPLOAD and
json_data[FIELD_TYPE] == TYPE_FILE:
                if json_data[FIELD_STATUS] == 200:
                    md5_recv = json_data[FIELD_MD5]
                if file_md5 != md5_recv:
                    file_delete(connection_socket, token,
key, username)
                    file_upload(username,connection_socket,
file_path, token)
                else:
                    print(json_data)
                    connection_socket.close()
                    return
```

Transforming the certain algorithms designed previously into realistic Python code is a tough task. We referred to the existing server-side code carefully to seek for clues and discussed with each other to address this difficult issue.

V. Testing and results

In this section, we conduct a series of tests on workability and performance of the application on two virtual machines.

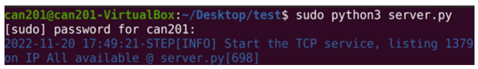
A. Testbed environment

Tests are conducted in the same host where implementation is finished. VirtualBox is used to create two virtual machines (VM1 and VM2) and the version is VirtualBox 6.1.40. Each VM has the same memory speed and CPU power as the physical host. However, the memory is 4 GB for each VM. Ubuntu-20.04 64-bit is the operating system installed in each

VM. Python 3 is also installed and used to test. The two VMs are totally identical since VM2 is created by cloning VM1.

B. Test steps and results

1) Functional testing: VM1 is used to simulate server and VM2 emulates client. We first run server-side code on VM1 and the blue log information appears on the terminal in the snapshot as shown in snapshot, which suggests that the bugs are successfully fixed.



```
can201@can201-VirtualBox:~/Desktop/test$ sudo python3 server.py
[sudo] password for can201:
2022-11-20 17:49:21-STEP[INFO] Start the TCP service, listing 1379
on IP All available @ server.py[698]
```

Then, a Word document of size approximately 300 KB is selected to upload as a test. Specify all the parameters required and run the code. Token is printed correctly on the first line of the terminal shown in the following snapshot, which indicates that the client gets authorization and manages to login to the server. The first line below the printed token returns the upload plan containing all the necessary information. The status code in each received packet is 200 and the last packet received returns md5 value, which shows that the implementation of file-uploading function succeeds.

```
can201@can201-VirtualBox:~/Desktop/test$ sudo python3 client.py --server_ip 192.168.56.101 --id 2036328 --f test_document.docx
[sudo] password for can201:
Token: MjAzNjMyOC4yMDIyMTEyMDA1MDQyMi5sb2dpbi4xNmRmMzliZWRhMmRhYmU2ZDI0MTI4ZTkyMzlmMTRiYQ==
{'key': 'test_document.docx', 'size': 303610, 'total_block': 15, 'block_size': 20480, 'operation': 'SAVE', 'direction': 'RESPONSE', 'status': 200, 'status_msg': 'This is the upload plan.', 'type': 'FILE'
}
{'key': 'test_document.docx', 'block_index': 0, 'operation': 'UPLOAD', 'direction': 'RESPONSE', 'status': 200, 'status_msg': 'The block 0 is uploaded.', 'type': 'FILE'}
{'key': 'test_document.docx', 'block_index': 1, 'operation': 'UPLOAD', 'direction': 'RESPONSE', 'status': 200, 'status_msg': 'The block 1 is uploaded.', 'type': 'FILE'}
{'key': 'test_document.docx', 'block_index': 2, 'operation': 'UPLOAD', 'direction': 'RESPONSE', 'status': 200, 'status_msg': 'The block 2 is uploaded.', 'type': 'FILE'}
{'key': 'test_document.docx', 'block_index': 3, 'operation': 'UPLOAD', 'direction': 'RESPONSE', 'status': 200, 'status_msg': 'The block 3 is uploaded.', 'type': 'FILE'}
{'key': 'test_document.docx', 'block_index': 4, 'operation': 'UPLOAD', 'direction': 'RESPONSE', 'status': 200, 'status_msg': 'The block 4 is uploaded.', 'type': 'FILE'}
{'key': 'test_document.docx', 'block_index': 5, 'operation': 'UPLOAD', 'direction': 'RESPONSE', 'status': 200, 'status_msg': 'The block 5 is uploaded.', 'type': 'FILE'}
{'key': 'test_document.docx', 'block_index': 6, 'operation': 'UPLOAD', 'direction': 'RESPONSE', 'status': 200, 'status_msg': 'The block 6 is uploaded.', 'type': 'FILE'}
{'key': 'test_document.docx', 'block_index': 7, 'operation': 'UPLOAD', 'direction': 'RESPONSE', 'status': 200, 'status_msg': 'The block 7 is uploaded.', 'type': 'FILE'}
{'key': 'test_document.docx', 'block_index': 8, 'operation': 'UPLOAD', 'direction': 'RESPONSE', 'status': 200, 'status_msg': 'The block 8 is uploaded.', 'type': 'FILE'}
{'key': 'test_document.docx', 'block_index': 9, 'operation': 'UPLOAD', 'direction': 'RESPONSE', 'status': 200, 'status_msg': 'The block 9 is uploaded.', 'type': 'FILE'}
{'key': 'test_document.docx', 'block_index': 10, 'operation': 'UPLOAD', 'direction': 'RESPONSE', 'status': 200, 'status_msg': 'The block 10 is uploaded.', 'type': 'FILE'}
{'key': 'test_document.docx', 'block_index': 11, 'operation': 'UPLOAD', 'direction': 'RESPONSE', 'status': 200, 'status_msg': 'The block 11 is uploaded.', 'type': 'FILE'}
{'key': 'test_document.docx', 'block_index': 12, 'operation': 'UPLOAD', 'direction': 'RESPONSE', 'status': 200, 'status_msg': 'The block 12 is uploaded.', 'type': 'FILE'}
{'key': 'test_document.docx', 'block_index': 13, 'operation': 'UPLOAD', 'direction': 'RESPONSE', 'status': 200, 'status_msg': 'The block 13 is uploaded.', 'type': 'FILE'}
{'key': 'test_document.docx', 'block_index': 14, 'md5': '6ce9fcb64fcc6bb1da997cbffd7dbfa0', 'operation': 'UPLOAD', 'direction': 'RESPONSE', 'status': 200, 'status_msg': 'The block 14 is uploaded.', 'type
': 'FILE'}
```

2) Average performance testing: Further, five different size of files for each of two different types are collected to test upload performance and the types are mp3 and docx. The detailed information is presented in Fig3. For the same file type, every different size of file is tested multiple times to calculate average performance and the results suggest that upload time is proportional to file size. For both types of file tested, the maximum file upload time is less than 0.16s, which indicates that the performance is efficient for relatively small file (less than 1 MB). As is illustrated in Fig3, the type of file may influence the transmission performance if the file size is within 1 MB. However, due to the rough evaluation method on uploading performance, it is hard to draw precise conclusion according to the statistics.

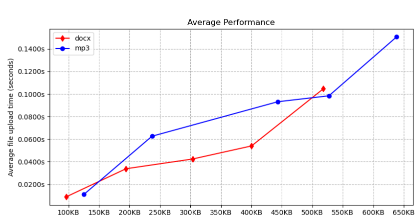


Fig. 3. Average performance of each file

In addition, the uploading speed of each block which can reflect the performance of each stage in the uploading process

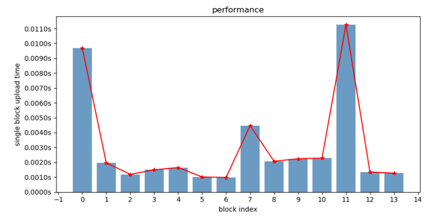


Fig. 4. Average performance of each block

is also a great criterion to evaluate performance. Fig4 shows that the majority of the upload speed in each block is less than 0.003s which means the upload performance of STEP protocol is relatively stable. There are some exceptions that the upload speed is higher than 0.003s. However, the maximum uploading time is less than 0.0110s, which indicates the high reliability of STEP protocol.

VI. Conclusion

File transmission has already popularized and is widely-used in people's daily life. In this report, we initially presented diagrams of the solution to the design and some exposition. Pseudo codes of core algorithms and the corresponding explanations were also offered. Furthermore, we provided the detailed information of implementation environment and real codes to implement core functions required in this application. The results were obtained after testing, showing the application developed is workable and the performance is high-efficiency. The protocol STEP actually serves more than file-uploading function. Data services and other file services can also be implemented by STEP. Besides, Client can only indicate one file to upload each time in this application. Therefore, the future work can focus on adding services, e.g., implement file download in the application or add function of uploading multiple selected files one time.

References

[1] C. Xiaojin, "Research on file upload based on html5," in *2014 11th International Conference on Service Systems and Service Management (ICSSSM)*, 2014, pp. 1–3.

[2] Y. Yubing and L. Zhanping, "Research of fast and safe large files transmission based on socket," in *2012 7th International Conference on Computer Science Education (ICCSE)*, 2012, pp. 483–485.

[3] H. Park, S. Lee, and Y. Shin, "High-speed transmission and control plan on high-definition video file using parallel tcp," in *2012 14th International Conference on Advanced Communication Technology (ICACT)*, 2012, pp. 1205–1208.

[4] H. Yao and S. Goto, "Performance evaluation of file transmission in content-centric network," in *2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems*, vol. 02, 2012, pp. 755–756.

[5] S.-Y. Chang, H.-T. Chiao, R.-K. Sheu, L.-C. Chen, and Welly, "Accelerating transmission of streaming files based on al-fec protection blocks," in *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, 2021, pp. 1188–1193.

[6] I. Malhotra, P. Chandra, and S. K. Majumdar, "Route optimization application using server-client architecture and google apis," in *2019 6th International Conference on Computing for Sustainable Global Development (INDIACom)*, 2019, pp. 210–214.