COURSEWORK 1 REPORT GUIDANCE

1st Jing Yin
ID number: 2034355

2nd Yumeng Zhang
ID number: 2035352

3rd Janis Anerauds
ID number: 2032936

***Abstract*—This project requires us to implement a client application of file uploading based on STEP protocol through python socket programming. Many mature file upload client programs have been implemented in the market. These programs achieve the primary file upload function. In addition to synchronous upload, offline transfer and other complex functions are also implemented. Referring to mature products, we designed our products. It also implements file upload between two computers based on the STEP protocol. During the test, we used files of different sizes for testing. The test environment is defined as two Linux virtual machines. The test results were pretty good, and we could upload files reasonably quickly, so we plan to develop a more robust product in the future.**

I. INTRODUCTION

A. background

File uploading has become essential to today's network management information system. However, the data type, length, compression method, and other information are opaque to the transport layer. Furthermore, the client itself knows the data transmission format because it is the one to transfer the data. However, the transmission side needs to learn the data transmission format. For example, if the client sends a JSON text, the Server does not know that it is JSON and not some other type of data. This is something we need to think about.

B. challenge

In this coursework, we first sorted out the protocol and client architecture. Then the C/S network architecture and the approximate structure of the client are designed. Finally, the functions of establishing the connection, executing authorization, getting tokens and uploading files were realized using the OOP programming idea and design diagram. In addition, we also performed functional tests and performance tests on the client application.

C. practice relevance

The project we completed this time also has some potential applications. The first would be simple data/file transfer from pc to Server, meaning people can use it as very primitive cloud hosting. Second, create a specific password system that expands the md5 hashing and implements more complex encryption. In that case, it can be used to store files on a global scale. It can be easily used by people in rural areas with a limited internet connection, as it splits the file into several packets, limiting the needed bandwidth.

Identify applicable funding agency here. If none, delete this.

D. contributions

In this project, we first combed through the protocol and client architecture. Then design c/s network architecture and client structure. Finally, using OOP programming ideas combined with the design diagram, the client can establish a connection, execute authorization, get tokens and upload file functions. In addition, we also carried out functional and performance tests on the client application.

II. RELATED WORK

Since the data information is opaque to the transport layer, this problem must be considered every time the data is transferred. Therefore, it is necessary to extract these generality problems and design a new protocol based on TCP/UDP. Therefore, we can use the corresponding application layer protocol to transmit application-specific data. There is no need to implement the application layer protocol again and transfer it to the transport layer. Instead, the data is sent to the application layer protocol, which calls the transport layer to complete the data transmission. For the application, we designed, as long as the data is generated, then sent to the application layer protocol for transmission, and finally received from the application layer protocol data sent from the peer end, directly processing.

Several application layer protocols have been designed for the implementation of file uploading. The Data Distribution Service(DDS) [1] is a data-focused protocol. In addition, Extensible Messaging and Presence Protocol(XMPP) [2], as an open communication protocol, can communicate asynchronously between two or more network entities in real-time. Hypertext Transfer Protocol (HTTP) [3], as a transport protocol, implements the transfer of Hypertext Markup Language (HTML) from the WEB server to the local browser. The dedicated Internet application protocol for constrained devices is the CoAP [4] protocol. This protocol allows restricted devices, called nodes, to communicate with the wider Internet. Furthermore, MQTT(Message Queue Telemetry Transmission) [5] is a messaging protocol. It can provide real-time, reliable messaging services and connect remote devices with very little code and limited bandwidth.

Regarding these protocols, we designed a file-uploading client application based on our STEP protocol. The STEP protocol is excellent:

It is a stateless protocol, meaning it can run at any time, on any supported platform, without needing server-side HDDS/SSDS or any other drives that require the prior state to exist. It opens as a new server each time it runs, so users do not need to set it up beforehand, and it allows for faster functionality.

It is very lightweight, which means users can run it on any possible server, and users do not need any specific hardware to run it.

Request/response allows for better error monitoring and better implementation. In the case of HTTP, a single error may mean many different errors, but in the case of STEP, they are limited and specified by the protocol.

It operates by authorization, limiting the possibility of possible connections and service interruptions and ruling out the possibility that someone could easily break in, especially with MD5 hashing, which can only be forced by a method that knows the correct encryption key.

III. Design

A. Server analysis

This section intends to analyze the Server's workflow following the STEP protocol. Then combined with the relevant knowledge of TCP connection, the C/S network architecture is designed. First, the Server creates a socket, binds the IP address and port, and specifies the number of connection requests the Server can accept simultaneously. The program then blocks until a client connects to the Server. After a TCP connection is established, the Server performs different operations based on the received data.

Firstly, analyze the Login function. The request format is shown in table I for the Server to perform a login operation. The client will, in turn, check the correctness of these messages and return an error status or a successful login signal. After successful login, the Server will send a response message containing the token.

Secondly, analyze how to get permission to upload files. This function corresponds to the save operation of file_process on the Server. The request format was shown in table II for the Server to jump to this operation. The Server will check the correctness of the information in turn and return the error status if any error occurs. The Server calculates the upload plan based on file size if all the information is correct. Finally, the information containing the key and upload plan is returned to the client, where the key is the upload permission.

Finally, analyze the function of uploading files. The request format should be shown in table III for the Server to jump to the file upload operation. The client will start receiving and saving the file block by block if all the information is correct. After receiving all the file blocks, the Server will calculate the md5 value of the entire file and return it to the client.

B. Design the C/S architecture and Client

The C/S network architecture we designed is shown in Figure 1. The client is connected to the Server through TCP protocol. The Server can process a maximum of 21 connection requests. The client application performs other functions, such as uploading files according to the STEP protocol. According to the analysis in 3.1, we designed the general workflow of C/S, as shown in Figure 2. The client first establishes a connection with the Server. The steps of authorization, obtaining the token, and uploading the file are then followed. Finally, the connection is

To determine the general framework, we designed the detailed architecture of the client, as shown in Figure 3. Using OOP programming ideas, we designed six bodies to do different jobs. The figure shows the functions of the six principals and the kernel algorithms they implement.

IV. Implementation

A. Implementation Steps

The host operating system used in our development is macOS, the CPU is a 2.3GHz Quad-Core Intel Core i7, and the memory is 16 GB. In addition, the primary IDE used for development is Virtual Studio Code. The implementation steps are mainly divided into four parts.

Firstly, we find the wrong statement in the Server and change it so that the Server can be set up typically. We found six syntax errors and added a python library.
Line 531 - change from variable data to variable json_data as it is used later in the while loop. We need to set it to none as there could be no JSON, and it would not set it to anything otherwise.
Line 646 - change from data_process to file_process as we need to call the correct function to call for file processing
Line 661 - change from SOCK_DGRAM to SOCK_STREAM since our connection is based on TCP, not UDP.
Line 671 - target change to the step_service(all lowercase)to call the correct service.
Line 673 - added th.start() to start the threading task
Line 693 - added () after main to call the function main
Line 16 - import hashlib since we need to use it during calculating md5 of the file.

Secondly, we set up the main framework of the client and established the connection with the Server according to the design of C/S in III-B. We rely on the socket(AF_INET, SOCK_STREA); from python socket library to create a socket with the type SOCK_STREAM because the connection is TCP-based. Then call socket.connect(Ip, port); to establish a connection with the Server. The Ip and port need to be obtained from the command statement, so we use the argparse library to create a method to parse the parameters. Finally, call socket.close(); to close the connection.

Thirdly, we start implementing the login function to get the token. Because the transmitted and received data formats are JSON and binary, we need first to implement the methods for constructing and retrieving packets, respectively. The JSON library is used to build and parse the JSON format information. There is no specific data type for dealing with bytes in python, so we need to use the struct library to handle bytes. Once the methods are set up, create a request using the format shown in table I and use socket.send(); to transfer data. The client will receive a response containing the token if the request is well-formed.

Fourthly, we implement upload application and file upload functions. Here we need to get the file path, so we import the os library. To get permission to upload, the client needs to construct a permission request, as shown in table II, where the token comes from the login section. Then send it to the Server. This is very similar to sending a login request. If the requested information is correct, the client will receive the

Type	AUTU
Operation	LOGIN
Direction	REQUEST
Username	username
Password	md5(username)

TABLE I
LOGIN REQUEST

Type	File
Operation	SAVE
Direction	REQUEST
Token	loginToken
Size	File size
Key	key

TABLE II
PERMISSION REQUEST

Type	File
Operation	SAVE
Direction	REQUEST
Token	loginToken
File	file_key
Block_index	block_index

TABLE III
UPLOAD REQUEST

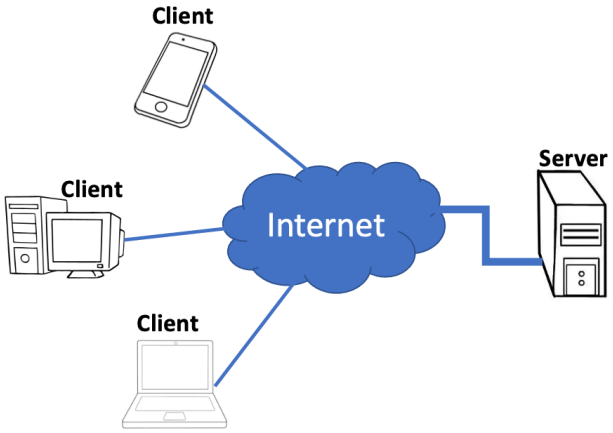

Fig. 1. The C/S network architecture

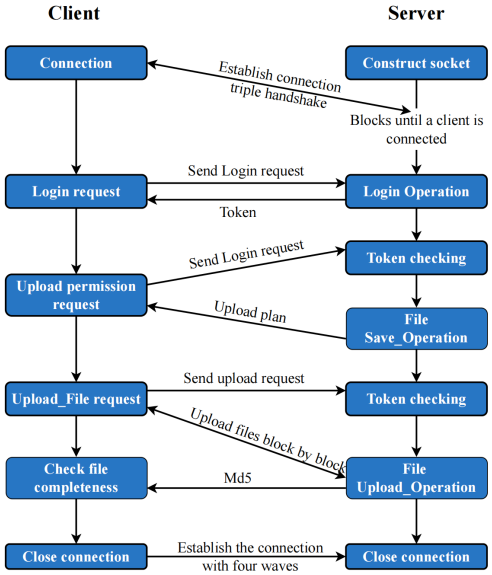

Fig. 2. The general workflow of C/S

response information, including key(permission) and upload plan. Finally, the file is split according to the upload plan, and the upload request is sent in a very similar process to the previous two. In this step, we need to precompute the md5 value of the file using the hashlib library. It is compared with the md5 sent back by the Server. If the two values are the same, the file upload is successful.

B. program flow charts

The workflow of the application performing authorization, obtaining tokens, and uploading files are shown in Figure 4. Client1 and Server1 complete the login. Client2 and Server2 completed the upload permission. Client3, Server3 and Slient3 complete the file upload, and Client4's primary role is to use md5 to confirm the integrity of the file upload. We have added extra functionality to client3, which is shown in red in the figure. The client mainly uses the key to avoid uploading the same file. However, different files with the same key could not be uploaded successfully. So, we designed a function to change the filename based on time automatically. When the client receives 402 status after permission, the client will automatically replace the key with the time plus key and re-apply for upload permission with the new key.

C. Programming techniques and difficulties

We adopt the method of designing the architecture first and completing the programming from abstract to concrete. The program is designed based on the idea of OOP. We first design six agents in III-B using abstract thinking and object-oriented ideas, which are responsible for different functions. Among them, the main body is responsible for implementing the main functions, and the other five principals are to ensure that the main body runs smoothly. In addition to that, we documented the implementation. This is very important in group work. This helps non-programmers understand the code quickly.

During the implementation, we encountered some difficulties. One of the biggest difficulties is mastering the overall architecture. It is hard to tease out a clear server-side structure because of the protocol's size and the server code's length. However, in the end, through clever abstraction, we could draw many diagrams and understand the protocol and Server. We use this technique later in the client build as well.

V. TEST

A. testing environment

The test host environment is the same as the implementation environment

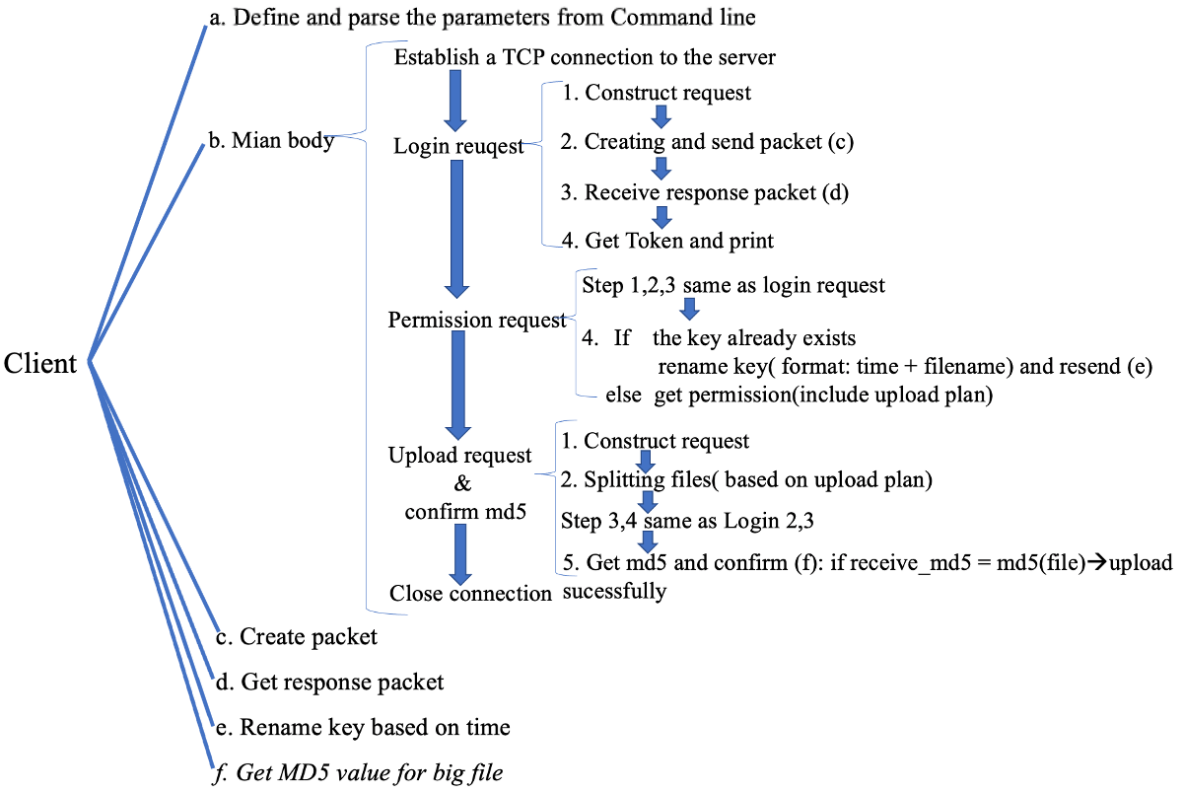

Fig. 3. The architecture of the client

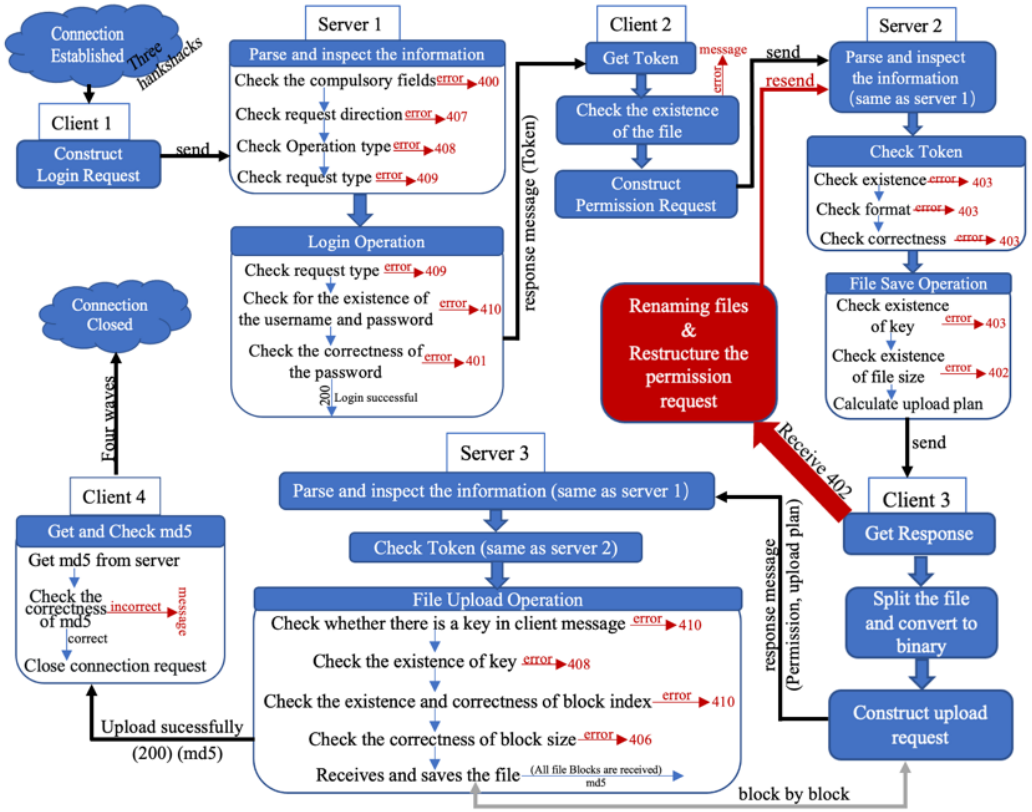

Fig. 4. The workflow of the application perform

B. testing steps

```
(base) can201@can201-VirtualBox:~$ sudo python3 client.py  -server_ip 192.168.1.107 -id 10001, -f ./file1.txt
Token: MTAwMDEsLjIwMjIxMTEzMjAwMjEzLmxvZ2luLmI3Y2IxOTk3NDM5ZmJjNzliMjY5MmViMWFlMmUyZWFl
{'key': '20221113200213-file1.txt', 'block_index': 0, 'operation': 'UPLOAD', 'direction': 'RESPONSE', 'status': 200
, 'status_msg': 'The block 0 is uploaded.', 'type': 'FILE'}
{'key': '20221113200213-file1.txt', 'block_index': 1, 'md5': '40c901b433d81c5b5bc0a2419159d1b1', 'operation': 'UPLO
AD', 'direction': 'RESPONSE', 'status': 200, 'status_msg': 'The block 1 is uploaded.', 'type': 'FILE'}
Server has received file properly, md5 match! MD5: 40c901b433d81c5b5bc0a2419159d1b1
Total upload time is 0.016054391860961914
```

Fig. 5. The sanpshot of test steps

C. testing results

spss curve fitting was used to select linear and quadratic curves respectively, with file size as the independent variable and file upload time as the dependent variable. The results are as follows:

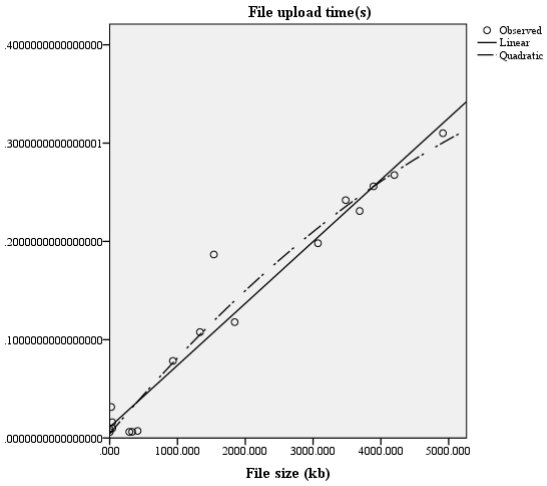

Fig. 6. Analysis of performance

Model Summary and Parameter Estimates
Dependent Variable: File upload time

Equation	Model Summary					Parameter Estimates		
	R Square	F	df1	df2	Sig.	Constant	b1	b2
Linear	.955	337.135	1	16	.000	.011	6.291E-5	
Quadratic	.960	181.631	2	15	.000	.004	8.132E-5	-4.289E-9

The independent variable is File size.

From the perspective of R squares after regression, the R squares of linear regression and quadratic regression are 0.955 and 0.96, respectively, which are both greater than 0.9, indicating that the two fitting methods have good fitting effect, and the significance of chi-square test is less than 0.01, indicating that the regression effect is significant.

The linear regression equation is:

$$\text{file upload time} = 6.291 \times 10^{(-5)} * \text{file size} + 0.011$$

The quadratic regression equation is:

$$\text{file upload time} = -4.289 \times 10^{(-9)} * \text{file size}^2 + 8.132 \times 10^{(-5)} * \text{file size} + 0.004$$

As seen from the trend chart of the fitting curve, there is an apparent positive correlation between and, with the increase of, there is a corresponding increase. In addition, the file upload program we completed achieves high efficiency in file uploading.

VI. CONCLUSION

We implemented a client application based on STEP protocol file upload in this project. Furthermore, this system provides efficient and stable uploads for files of different sizes. The client establishes a connection, executes authorization, obtains a token, and uploads a file. The tasks are processed one by one. In this way, the system can quickly and successfully upload files. In the future, we will simplify the design of the code and increase its readability of the code. In addition, we will take a few more steps to encrypt the code to improve security.

REFERENCES

[1] Z. Abdellaoui and S. Hasnaoui, "Dds middleware on top of flexray networks: Simulink blockset implementation of electrical vehicle using flexray protocol and its adaptation to dds concept." *Soft Computing: A Fusion of Foundations, Methodologies and Applications*, vol. 23, no. 22, pp. 11 539 – 11 556, 2019. [Online]. Available: https://login.ez.xjtlu.edu.cn/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edssjs&AN=edssjs.2185EC33&site=eds-live&scope=site

[2] M. A. Aftab, S. S. Hussain, T. S. Ustun, and A. Kalam, "Iec 61850 communication and xmpp based design of virtual power plant," in *2021 31st Australasian Universities Power Engineering Conference (AUPEC)*, 2021, pp. 1–5.

[3] Q. Hu, M. R. Asghar, and N. Brownlee, "A large-scale analysis of https deployments: Challenges, solutions, and recommendations." *Journal of Computer Security*, vol. 29, no. 1, pp. 25 – 50, 2021. [Online]. Available: https://login.ez.xjtlu.edu.cn/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=bsu&AN=148596209&site=eds-live&scope=site

[4] L. Yinliang, L. Jinyong, Y. Zhiyong, X. Qiyuan, X. Quan, and Y. Lei, "A coap-based communication mapping method for power distribution internet of things." *2022 China International Conference on Electricity Distribution (CICED), Electricity Distribution (CICED), 2022 China International Conference on*, pp. 243 – 248, 2022. [Online]. Available: https://login.ez.xjtlu.edu.cn/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.9928955&site=eds-live&scope=site

[5] M. Carratu, V. Gallo, and V. Paciello, "Ieee 1451: Communication among smart sensors using mqtt protocol." *2022 IEEE International Symposium on Measurements & Networking (M&N), Measurements & Networking (M&N), 2022 IEEE International Symposium on*, pp. 1–6, 2022. [Online]. Available: https://login.ez.xjtlu.edu.cn/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.9887708&site=eds-live&scope=site