Xi'an Jiaotong-Liverpool University
西交利物浦大學

CAN201 Coursework Report

Gaojie Hou (2144673), Zihan Deng (2144196),
Chengze Liu(2142808), Ruixuan Chen (2144955), Jinai
Ge (2035693)

November 15, 2023

1 Abstract

This project presents a client-server architecture leveraging the TCP protocol to enable secure and efficient file exchange. The client application provides users with the ability to log in and obtain an access token for server interactions. Users can upload files from their local system, with the client segmenting files into discrete blocks for transmission. The server undertakes a rigorous validation process, ensuring the integrity of the whole file and subsequently generating a unique key and MD5 hash for the uploaded file upon successful upload completion. The client application features a user-friendly progress bar, offering real address issues, such as login failures, upload errors, or file integrity checks. This project is a testament to a reliable, and efficient means of interacting with server resources for seamless file sharing while upholding data integrity.

2 Introduction

With the continuous development of technology, the transmission and storage of data have become increasingly important. In the face of this challenge, this project aims to develop a reliable file transfer system based on a custom STEP(The Simple Transfer and Exchange Protocol) which is a reliable TCP-based protocol. This protocol establishes a data transmission standard to ensure data integrity. The project involves both server-side and client-side components, with the server side responsible for handling incoming requests and the client side responsible for initiating requests and performing file transfers. By implementing this file transfer system, we will address the integrity issues in traditional file transfers, thereby improving the reliability of data transmission.

3 Related Work

JSON is a lightweight key-value style data exchanging format. JSON has a simple and very easy-to-learn syntax, with readable output as long as it includes whitespace in the form of indentation and line breaks[1]. The TCP protocol is designed to reduce its sending rate when congestion is detected. TCP can provide a virtual circuit (connection-oriented) communication service across the network. Rules like flow control, and error correction are also included in TCP [2]. HTTP follows the receiver-driven pattern. A client should send an HTTP request with a URL to get an HTTP response back. The STEP required in this coursework is located in the application data of TCP, while TCP is placed in the data area of IP. The outcome of the coursework is relevant to storing data in the server whose potential application is cloud storage which requires high availability, reliability, and data consistency. As for the security of cloud storage, data can be prevented from accidental erasure or hardware crashes as there are continuous copies of data[3].

4 Design

4.1 Server Side: Debug

Task 1 is about debugging a server-side program that implements the STEP protocol and a client-server architecture for operations like file processing. The server listens for client requests on port 1379 using TCP. However, there's an issue in the ‘`tcp_listener()`‘ function where the socket type needs to be changed to STREAM for stable data transmission over TCP, and missing parameters for creating threads should be added.

To use the file cloud processing service you need to log in first. Upon successful login, the server checks the client's requested operation type and invokes corresponding methods. There's a bug related to the output string 'server is ready' in this function.

For data transmission, original content must be encoded into binary data due to the lower-level network link's binary data requirement. Additionally, handling conversions between dictionaries and JSON is essential when transmitting key-value pairs. It is worth noticing that in the file process function, the he operation identifier of the sent packet is incorrect and has been modified.

Algorithm 1 File Block Selection

```
if $index != total\_block - 1$ then
    $final\_file \leftarrow send\_file[MAX\_FILE\_SIZE \cdot index : MAX\_FILE\_SIZE \cdot (index + 1)]$
else
    $final\_file \leftarrow send\_file[20480 \cdot index :]$
end if
```

4.2 Client Side: Workflow

4.2.1 Authentication

To connect with the server, the client initiates a "LOGIN" request, obtaining a token in response to subsequent operations. This involves the user providing parameters, which require parameter-passing functionality. Data conversion methods similar to those on the server side are essential. After a successful login, the user's token is extracted from the server's response and displayed.

4.2.2 File Upload and Storage

Task 3 involves uploading files to the server while ensuring integrity. The server provides an upload plan based on STEP, dividing the file into blocks. Algorithm 1 demonstrates the client's simple operation for file chunking. It is worth noting that in the STEP definition, the operations related to upload planning and sending/receiving are not implemented by UPLOAD but by SAVE. The client requests the SAVE operation, deciding whether to create a custom key or use a server-generated key. After that, the client receives an upload plan, which is used to transfer the file, displaying progress. The server responds with the file's MD5 value, which the client compares with the local

MD5 for integrity verification. What's more, keeping the user informed of progress and the key's value is crucial.

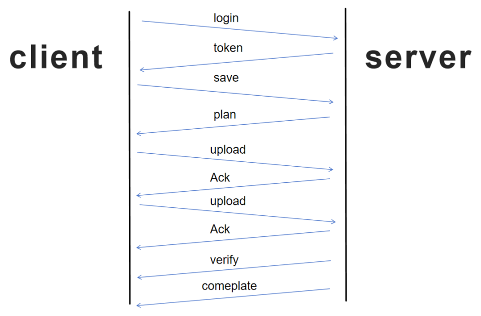


Figure 1: C/S communication

5 Implementation

5.1 Development Environment and Tools

The operating system of the computer is Windows 10, with Intel Core i5-11400U CPU and 16GB RAM. The program was developed on the Windows 22H2 operating system using Python 3.11 and *JetBrain* Pycharm as the IDE.

5.2 Implementation Process

The functions mentioned above are reused due to their functional similarities. The various client functionalities are implemented following these steps:

1. Initialization of sockets and variables is performed before the login operation.

2. The login operation is executed. The login requirements are converted into JSON format and transmitted in binary form. Subsequently, the program determines the success of the login by checking if the token information is present in the server's response packet.

3. Considering the save and upload operations, the basic principles are similar to the login operation. In both cases, data and information are packaged and the program waits for useful information in the response packet. If the program runs normally, the SAVE operation will return the number of file chunks in the upload plan. It is important to note that if any abnormal states occur during this process, the program will catch exceptions and prompt the user to retry.

4. After successful login and save operations, the first socket is closed. To enhance transmission performance, we utilize multithreading for the UPLOAD operation. Multithreading is implemented by concurrently creating a series of sockets, with each socket handling a block. Within

this context, semaphores are used to prevent thread conflicts and excessive waiting. Upon successful completion of the upload and verification of file integrity, the program returns the necessary information and exits.

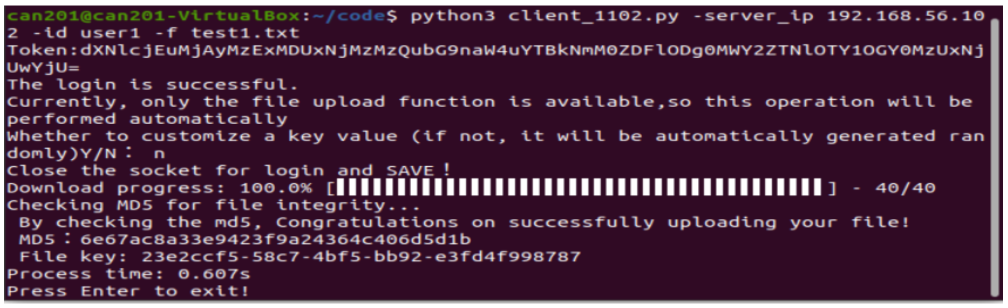
6 Testing and Results

The tests will be performed on two virtual machines with Ubuntu installed, one running the server program and the other running the client program. Each VM is assigned 4096MB of RAM.

1. Run the server program on one of the virtual machines and wait for the server side to start successfully. The specific steps are to open a terminal in the path where the server.py file is located, and enter the following command ”python3 server.py”.

2. Run the client program on the other virtual machine. In this test, a total of five files will be tested, all of which are `.txt` files containing randomly generated numbers from 0 to 9. Their sizes are 20,000 bytes (19.5kb), 40,000 bytes (39kb), 60,000 bytes (58.5kb), 80,000 bytes (78.1kb), and 100,000 bytes (97.6kb), respectively. Normally, after entering the command in the terminal, the terminal will output a prompt asking the user whether to customize the key value or generate it automatically. The specific steps are to open a terminal in the path where the client.py file is located and enter the following command `python3 -client.py --server_ip xxx.xxx.xxx.xxx --id xxx --f xxx`

3. To minimize errors, no other operations were performed on the test computer during the upload process. It can be seen that there is a linear relationship between the size of the file and the time taken to upload it. Figure 2 illustrates the terminal, while Figure 3 shows the performance of the uploading system.



```
can201@can201-VirtualBox:~/code$ python3 client_1102.py -server_ip 192.168.56.10
2 -id user1 -f test1.txt
Token:dXNlcjEuMjAyMzExMDUxNjMzMzQubG9naW4uYTBkNmM0ZDFlODg0MWY2ZTNlOTY1OGY0MzUxNj
UwYjU=
The login is successful.
Currently, only the file upload function is available,so this operation will be
performed automatically
Whether to customize a key value (if not, it will be automatically generated ran
domly)Y/N： n
Close the socket for login and SAVE！
Download progress: 100.0% [████████████████████████████████████████] - 40/40
Checking MD5 for file integrity...
 By checking the md5, Congratulations on successfully uploading your file!
 MD5：6e67ac8a33e9423f9a24364c406d5d1b
 File key: 23e2ccf5-58c7-4bf5-bb92-e3fd4f998787
Process time: 0.607s
Press Enter to exit!
```

Figure 2: Demonstration of the terminal

7 Conclusion

This project has successfully implemented a client-server architecture based on the TCP protocol, facilitating efficient file exchange. The developed client application enables users to log in and

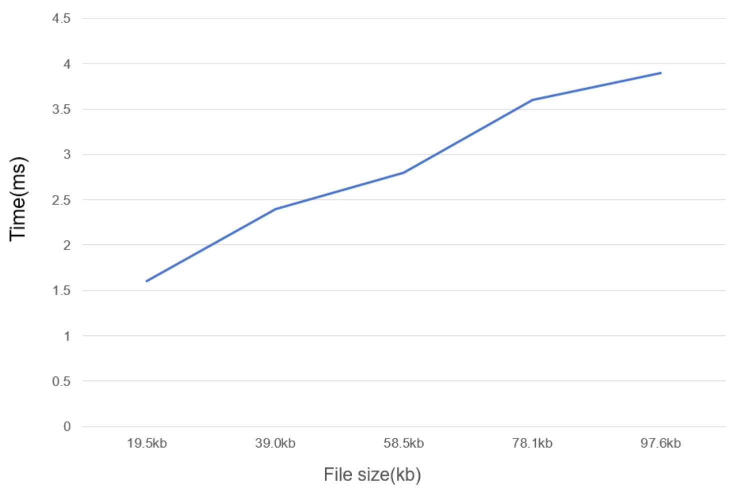


Figure 3: Test result

acquire access tokens for interacting with the server. In the implementation process, there are several issues worthy of discussion. When packaging data, we encountered the need to convert the data correctly due to the nature of TCP's stream sockets, and thus, it was necessary to select the right data structure and perform effective conversion. We found that only dictionary objects could handle such operations. Future work may involve optimizing the server's capacity to handle larger files, enhancing the system's scalability and robustness. Moreover, the project's security aspect needs reinforcement through enhanced user authentication mechanisms and the implementation of encrypted communication to strengthen security.

8 Acknowledgement

Gaojie Hou (2144673) contributes 20% to the project, Zihan Deng (2144196) contributes 20% to the project, Chengze Liu (2142808) contributes 20% to the project, Ruixuan Chen (2144955) contributes 20% to the project, and Jinai Ge (2035693) contributes 20% to the project.

References

[1] M. Eriksson and V. Hallberg, "Comparison between json and yaml for data serialization," *The School of Computer Science and Engineering Royal Institute of Technology*, pp. 1–25, 2011.

[2] G. C. Kessler, "An overview of tcp/ip protocols and the internet," *InterNIC Document, Dec*, vol. 29, p. 42, 2004.

[3] J. Wu, L. Ping, X. Ge, Y. Wang, and J. Fu, "Cloud storage as the infrastructure of cloud computing," in *2010 International Conference on Intelligent Computing and Cognitive Informatics*, pp. 380–383, 2010.