A STEP Protocol-Based Client-Server System for Reliable File Upload and Download

1st Jiahao.Qi
School of Advanced Technology
Xi'an Jiaotong-Liverpool University
Suzhou,China
ID

2nd Shuobai.Chen
School of Advanced Technology
Xi'an Jiaotong-Liverpool University
Suzhou,China
ID

3rd Zhenxi.Chen
School of Advanced Technology
Xi'an Jiaotong-Liverpool University
Suzhou,China
ID

4th Antian.Sun
School of Advanced Technology
Xi'an Jiaotong-Liverpool University
Suzhou,China
2362202

***Abstract*—This report presents the design and implementation of a C/S file transfer system based on the STEP protocol. The project addresses three key tasks:debugging and setting up the server,implementing token-based authentication, and performing file upload with integrity verification with MD5 checksum. The system supports concurrent client connections through multi-threading and ensures secure data transmission through structured packet formatting and error handling. This work demonstrates a practical application of network protocols in building reliable file transfer systems, with potential applications in educational and enterprise environments that require secure and efficient data exchange.**

I. Introduction

A. Task Specification

In the context of the information age, reliable data transmission is of vital importance. The client/server (C/S) model, centered on socket programming and protocols such as TCP, is precisely the cornerstone of modern network systems. This model supports many classic network applications, such as Telnet, FTP, Rlogin, and SMTP, all of which rely on the reliable connection, data integrity, and flow control mechanisms provided by TCP [1]. This project aims to develop a file upload and download client based on Python following the STEP protocol under this background, in order to build a fully functional C/S system.

B. Challenge

- Protocol Interpretation: Correctly implementing a custom application-layer protocol.
- Block-wise Transfer: Reliably uploading files in chunks based on a server-defined plan.
- Token Management: Securely handling the authentication flow and session token.

C. Practice Relevance

Considering the characteristics of the STEP protocol, it is highly suitable for the following scenarios: file transfer that requires authentication, small and medium-sized concurrent user environments, and scenarios that need to transfer logs and audit trails. In conclusion, this is a highly practical, stable, and reliable file transfer solution for small and medium-sized enterprises, especially suitable for environments such as enterprises, schools, and institutions that require secure and controllable file exchange. It has great application potential in this regard.

D. Contribution

Key contributions of this project include:

- Complete implementation of the STEP protocol: Handle message formats and operation types strictly in accordance with protocol specifications.
- Security Authentication System: It has implemented token-based authentication and session management mechanisms.
- Reliable file transfer: A block transfer mechanism has been developed and data integrity is ensured through MD5 verification.
- Comprehensive error Handling: A complete error status code system has been implemented, enhancing the system's robustness.
- Server Optimization: Fixed key bugs to ensure stable server operation.
- System Architecture: Build a complete C/S solution that supports multi-user isolation

II. Related Work

Network traffic redirection encompasses techniques that optimize data transmission paths to enhance network performance. In Passive Optical Networks, Hwang and Liem [2] demonstrated local traffic redirection among ONUs that reduces OLT bandwidth consumption through their REDIRECT DBA scheme. Similarly, Huang et al. [3]implemented application-aware redirection in 5G MEC environments, dynamically routing user requests to edge servers based on throughput thresholds.

Our implementation of the STEP protocol represents a fundamental embodiment of these redirection principles. Although operating on a simpler scale, our system employs core redirection mechanisms: socket-based path control between client and server, chunked transmission for traffic management, and token-based access control.

The technology evolution path clearly delineates our foundational work from more advanced traffic redirection systems. Our C/S transmission establishes the essential protocol-level groundwork, whereas the cited research represents sophisticated implementations in specialized network architectures. This progression not only highlights our project's role in paving the way for complex traffic redirection technologies, but also reveals the possibility of more network traffic redirection applications

III. Design

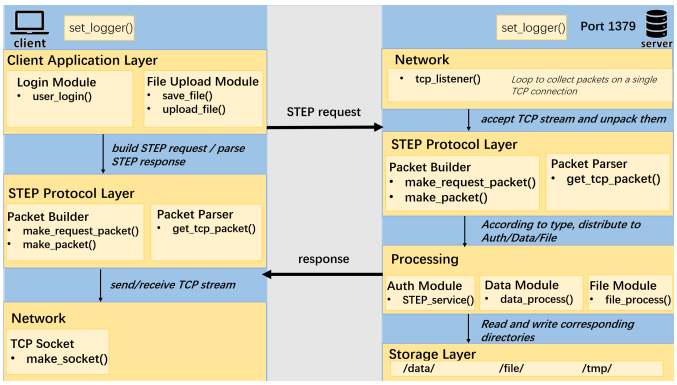

Fig. 1. Architecture diagram

The above diagram illustrates the architecture of both the client and the server. The client contains three main parts, which are the Client Application Layer, the STEP Protocol Layer, and the Network. The Client Application Layer is equipped with a Login Module, which is in charge of the part of authentication, and a File Upload Module, which has the function of requesting upload plans from the server and uploading files to the server. The STEP Protocol Layer provides the function of packing and sending/receiving information. The Network ensures the connection between the client and the server. As for the server side, it consists of the Network, STEP Protocol Layer, and Processing and Storage Layer. The Network runs a listener that waits for clients' connections. The STEP Protocol Layer performs the same function as the STEP Protocol Layer on the client side. The portion of Processing is made up of the Auth Module, the Data Module, and the File Module that processes three distinct types of requests, respectively. And the last part is the Storage Layer that stores the data and files uploaded from the clients, and the temporary file produced during the procedure of uploading.

The following workflow illustrates the interaction between the client, server, and the database. It represents the procedure of authorization, token fetching, and file uploading. The process starts with the client requesting to login, the server

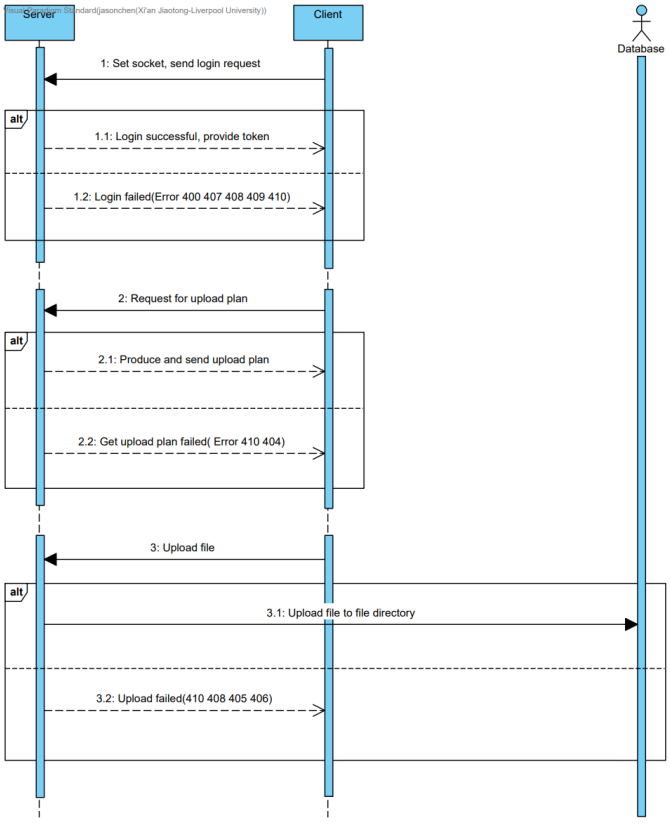

Fig. 2. Workflow

generates and returns an MD5 encoded token if the login is successful. Then, the client sends a request for an upload plan containing the token to the server, the server verifies the token, and returns an upload plan with key, file size, total block, and block size if the client's request is valid. Lastly, the client uploads a file with a token, the server verifies the token, and saves the file in the 'file' directory under the 'username' directory if the token and the uploaded file are confirmed.

Algorithm 1: Authorization — Client

```
if FIELD_TOKEN ∈ json_data then
    token ← json_data[FIELD_TOKEN]
    if token is None then
        raise Exception
    end
    else
        Log "Login Successful, the token is {token}"
    end
end
else
    Log "Login Failed, the token is not found!!"
end
```

Algorithm 2: Authorization — Server

```
if hashlib.md5(json_data[FIELD_USERNAME].
  encode()).hexdigest().lower() ≠
 json_data['password'].lower() then
    send(
     make_response_packet(OP_LOGIN,
     401, TYPE_AUTH, ''Password error
     for login.'', {}))
    return
end
```

Algorithm 3: File Uploading — Client

```
for i ← 0 to total_block−1 do
    bin_data ← f.read(MAX_PACKET_SIZE)
    json_data ← {FIELD_KEY :
     key, FIELD_BLOCK_INDEX : i}
    message ←
     make_request_packet(OP_UPLOAD,
     TYPE_FILE, json_data, bin_data)
    client_sock.send(message)
    recv_message ←
     client_sock.recv(MAX_PACKET_SIZE)
    upload_data, _ ←
     get_tcp_packet(recv_message)
    Log upload_data[FIELD_STATUS_MSG]
    UPLOAD_RESULT.append(upload_data)
    realtime_updating(i, total_block,
     start_time)
end
client_sock.close()
```

Algorithm 4: File Uploading — Server

```
file_path ← join('tmp', username,
 json_data[FIELD_KEY])
file_size ← getsize(file_path)
block_size ← MAX_PACKET_SIZE
total_block ← ⌈ file_size / block_size ⌉
block_index ←
 json_data[FIELD_BLOCK_INDEX]
fid.seek(block_size · block_index)
fid.write(bin_data)
fid.write(block_index + ''\n'' )
fid ← open(file_path + ''.log'', ''r'' )
lines ← fid.readlines() ; fid.close()
rval ← {FIELD_KEY : json_data[FIELD_KEY],
 FIELD_BLOCK_INDEX: block_index}
if len(set(lines)) = total_block then
    md5 ← get_file_md5(file_path)
    rval[FIELD_MD5] ← md5
    os.remove(file_path + ''.log'' )
    shutil.move(file_path, join('file',
     username, json_data[FIELD_KEY]) )
end
send( make_response_packet(OP_UPLOAD,
 200, TYPE_FILE, "The block {block_index} is
 uploaded.", rval) )
```

IV. Implementation

A. Development Environment

TABLE I
Development Environment

Field	Description
Operating System	Microsoft Windows 11
CPU	Intel(R) Core(TM) i7-14650HX
RAM	32G
IDE	JetBrains PyCharm Professional
Python Version	Python 3.14
Python Library	'argparse', 'shutil', 'json','os', 'math', 'socket', 'logging', 'struct', 'hashlib', 'time', 'threading'

B. Step of implementation

1) Determine the task requirements and divide the tasks.
2) Correct errors in the server program.
3) Write the client program based on the server program to implement the functions of authorization and file upload.
4) Run the program, conduct tests on different computers, and write the final report.

C. Programming Skills

- **Program modular design**: The entire application is evenly divided into multiple specific Python functions based on functionality, which enhances the readability and maintainability of the code.
- **Threading for Concurrency**: The program adopts a multi-threaded architecture to achieve efficient concurrent request processing. The server creates an independent worker thread for each client connection to ensure that the main listening loop continues to run uninterrupted, providing reliable support for multi-user concurrent operations.
- **Token-Based Authentication**: The program has established a security verification mechanism, ensuring system access security and data integrity through Token identity authentication and MD5 file hashing.

D. Actual Implementation

a) Authorization: The authorization starts with the client sending a login request consisting of option as LOGIN, type as AUTH, and payload containing username and password, which is generated by encoding the username and computing the MD5 digest. Then, if the request is valid, the server returns a token, which is oriented by replacing '.' in the username of the JSON data received from the client with '_', then combining the username with the current time and a static salt value, after that, computing the MD5 hash of the string and add it to the tail of the string, finally, Base64 encode the string. The client gets the token from the reply message of the server and includes it in the subsequent requests for file uploading.

b) File uploading: The part of file uploading can be divided into upload plan fetching and file uploading. As a starter, the client sends a request containing the option as SAVE, type as FILE, file size, and a field key denoting the file name. After receiving the request, the server will create an upload plan consisting of the key, the size of the file, the total block number, and the block size. Then the upload plan will be stored in the 'tmp' directory and returned to the client; this is how the client gets the upload plan. After that, each time the client reads a segment of the file to be uploaded, constrained by the block size, it packs it with the required protocol format and sends it to the server. The server writes the blocks received from the client into the 'tmp' directory, then moves the reconstructed file to the 'file' directory. At last, the server sends a reply to the client denoting the completion of file uploading. The whole process is logged by both the client and the server separately.

E. Difficulties And Solutions

One of the most challenging parts of the project is designing a security authentication system. We need to have thorough understanding of the MD5 algorithm's characteristics, and apply it to file integrity verification. We have understood the composition and function of quadruple and ensure transmission security through Base64 coding.

When implementing the function of reliable file transmission, we are faced with the challenge of data integrity. By designing a block transmission mechanism, files are divided into fixed-size blocks and uploaded. Combined with block index verification, temporary file status management(tmp), and MD5 hash verification on the server side, the integrity and recoverability of files during the transmission process are ensured.

V. Testing and Results

A. Test environment

The coding test were conducted on two laptops acting as the client and server respectively, which are connected to a hotspot provided by a smart phone. The following are the configuration of two laptops:

TABLE II
Test environment of Client

Field	Description
Operating System	Microsoft Windows 11
CPU	Intel(R) Core(TM) i7-14650HX
RAM	32G
Python Version	Python 3.14

TABLE III
Test environment of Server

Field	Description
Operating System	Microsoft Windows 11
CPU	Intel i9-13900H 13th
RAM	32G
Python Version	Python 3.12

B. Testing steps (Functional Test)

1) Server setup and debugging: For Task 1, the server program is first debugged and launched on the server laptop. The server prints log messages "Server is ready!" and "Start the TCP service, listening 1379 on IP All available", indicating that it is correctly bound and listening for incoming connections.

```
2025-11-12 19:33:03-STEP[INFO] Server is ready! @ server_windows.py[690]
2025-11-12 19:33:03-STEP[INFO] Start the TCP service, listing 1379 on IP All available @ server_windows.py[691]
```

2) User login and token acquisition: For Task 2, the client program is executed on the second laptop, which acts as the client. The program prints a log message such as "Login Successful, the token is ...", confirming that the login step is completed successfully and that the token can be used to authorize subsequent file operations.

```
2025-11-14 14:43:54-STEP_Client[INFO] uploadpython client/client_windows3.py --server_ip "192.168.0.22" --id "XiaXi" --f "D:\V3-S1\CAN201\Coursework_Part1\client\random_text_10MB.txt" 1>
```

```
2025-11-12 19:33:86-STEP_Client[INFO] Login status: 200 @ client_windows.py[157]
2025-11-12 19:33:86-STEP_Client[INFO] Login Successful, the token is WGlnNGkwMjAyNTExMTIxOTMzMDYubG9naW4uNDc1YTkyZGI4ZTQ3ZjYxMjNjNGYzMTYwNzkwNmZiODk
```

3) File upload using STEP: After obtaining the authorization token, the client invokes the SAVE operation by sending a message. The server responds with an upload plan including a key, the total number of blocks, and the block size. While uploading the file, for each block, the server replies with a status code indicating whether the block has been correctly received. Once all blocks have been uploaded, the server computes the MD5 checksum of the file and returns it to the client for integrity verification. The uploaded file is saved under the directory of the 'file'.

```
Uploading block: 2/3 [================------]100.0% Avg.Speed: 0.4MB/s ETA: 0.00s @ client_windows.py[104]
2025-11-12 22:14:59-STEP_Client[INFO] uploading cost time: 0.10402 seconds @ client_windows.py[221]
2025-11-12 22:14:59-STEP_Client[INFO] local md5:  ea2506567a68a4f246ac377cc4398a32 @ client_windows.py[225]
2025-11-12 22:14:59-STEP_Client[INFO] server md5: ea2506567a68a4f246ac377cc4398a32 @ client_windows.py[226]
2025-11-12 22:14:59-STEP_Client[INFO] MD5 check passed: file uploaded correctly. @ client_windows.py[228]
```

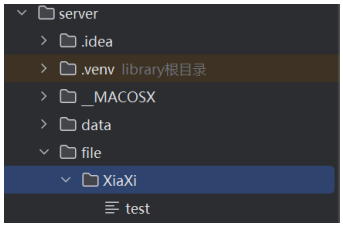

Fig. 3. File saved

C. Test Result

To evaluate the performance of the implemented file transfer system, several experiments were carried out on the two-laptop setup described above.

A set of test files with different sizes (5 MB, 10 MB, 20 MB, and 50MB) were prepared. For each file size, the upload operation was repeated 3 times, and the total uploading time was recorded for every run. Finally, we computed the average time to represent them respectively.

The testing result are shown below:

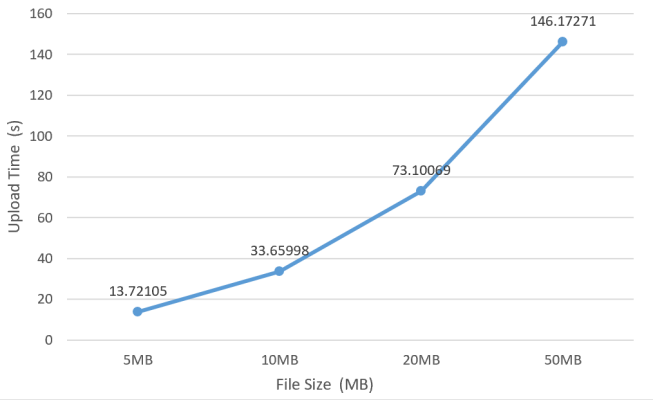

Fig. 4. Upload time for different sizes of files

Based on the data demonstrated above, it is observed that the total uploading time increases with the file size, as expected for a system that sends the file over a TCP connection in fixed-size blocks. Overall, the experiments suggest that the implemented STEP-based system can reliably transfer files between the client and server within the local network.

VI. Conclusion

The project achieved a TCP-based STEP protocol and its corresponding client/server system to support logging, data packing, authorization and token fetching, and file uploading. There are a series of significant issues concerning data transferring, such as data integrity, privacy, and transmission efficiency. The coherence of the transfer of large files has been a challenge in the field of networking; this is enhanced in the project by applying chunked transfer, by dividing the file into blocks to avoid the difficulty of transferring the file as a whole. The problem of incomplete files exposed is tackled by writing blocks into a temporary directory, then transferring the integrated file to the directory storing the upload files. Also, this project provides multi-client concurrency, which caters to the actual requirement of the current development environment. By having the above characteristics, this project can be implemented in various software development scenes. For instance, it can be implemented in a coursework submission system in universities, since the properties of this project satisfy the needs of multi-user uploading and large file transferring. The security is also ensured as the project provides authorization through a token, making it fit the requirements of cloud storage within the enterprise. To actually put this project into production, a few flaws can be improved. Such as adding the function of resumable upload, which can save bandwidth or time and efficiently recover after failures. In addition, implementing parallel multiple uploads can provide a much higher throughput for large files. Moreover, quotas and rate limiting can be a sufficient approach to preventing one tenant from overwhelming the system.

Acknowledgment

TABLE IV
INDIVIDUAL CONTRIBUTION PERCENTAGE

Name	Percentagen
Jiahao.Qi	25%
Shuobai.Chen	25%
Zhenxi.Chen	25%
Antian.Sun	25%

References

[1] M. Xue, and C. Zhu, "The socket programming and software design for communication based on client/server," presented at the Pacific-Asia Conference on Circuits, Communications and System, May. 16-17, 2009.

[2] I. Hwang and A. T. Liem,"A multimedia services architecture supporting local traffic redire in passive optical network," presented at the 11th International Conference on Optical Communications and Networks, Nov. 1-4, 2012.

[3] S. Huang, Y. Luo, B. Chen, Y. Chung and J. Chou, "Application-aware Traffic Redirection:A Mobile Edge Computing Implementation toward Future 5G Networks," presented at the IEEE 7th International Symposium on Cloud and Service Computing,Nov. 17-23, 2017.