Xi'an Jiaotong-Liverpool University

School of Advanced Technology

Networking Project on Simple Transfer and Exchange Protocol (STEP) v1.0

Authors:	*Student IDs*
Zheyuan Cao	2141805
Yanqing Fei	2143430
Ruiqi Chen	2141794
Fei Du	2141378
Yifan Hong	2144280

A Report Submitted For

CAN201 Introduction to Networking

November 13 2023

Abstract

File transmission plays a fundamental role in network communication. In this project, with the aim of achieving reliable and efficient data exchange between entities, socket programming is utilized to actualize a client-server file transfer system rooted on a TCP-based protocol.

1 INTRODUCTION

As the internet usage surges, file transfer remains a prevalent and significant aspect of network communication. Protocols establish standard rules for data transmission, dictating message format, transfer sequence, and associated actions. TCP, characterized by its reliability, flow control and connection-oriented nature, stands out in this context. This project employs Python Socket programming with a TCP-based protocol: Simple Transfer and Exchange Protocol (STEP), to fulfil three missions: debugging server codes, securing server authorization and accomplishing a client-server file upload. Specifically for contributions, rectifying server syntax errors enables token retrieval via client login, resulting in subsequent block-level file uploading. Challenges exist during implementation, including handling missing packages, calculating the password, and optimizing message block transfer algorithms. The eventual application offers streamlined communication with a cost-efficiency trade-off, potentially suitable for file management in small-scale or domestic settings. The project details are outlined as follows.

2 RELATED WORK

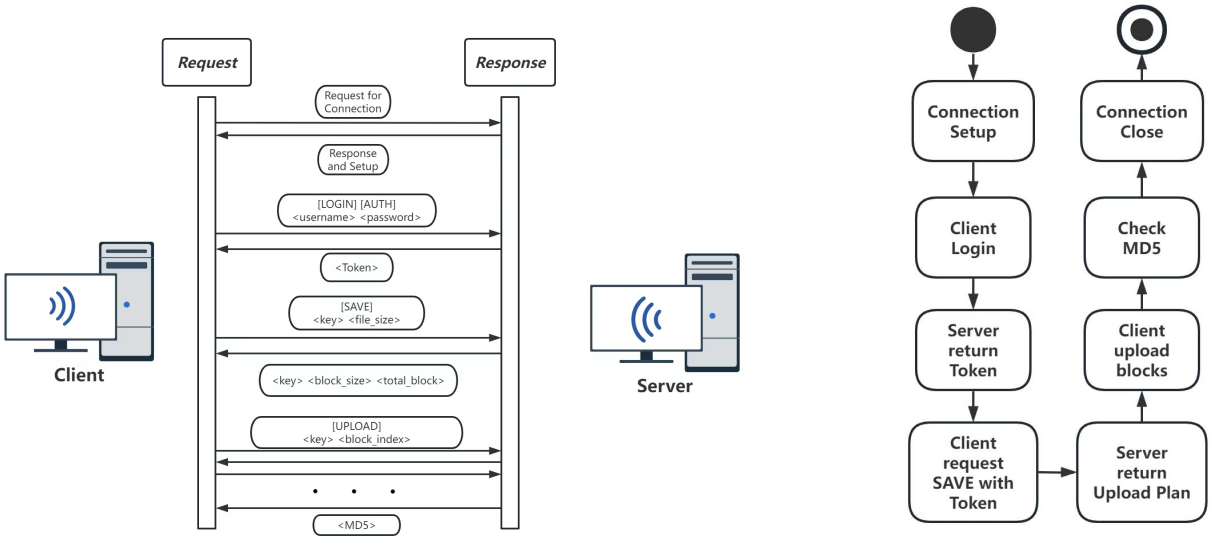
Network transmission, a key focus in scholarly research, necessitates a thorough preliminary analysis prior to commencing the work. Central to this domain are security and transfer [1]. Effective security measures, such as encryption algorithms and authorization techniques [2], guarantee the transfer reliability, aligning with the objectives of this project. Moreover, diverse factors can exert influence on transmission efficiency such as file size distribution arrangement and network stability [3]. Therefore for file transfer specific to this task, efficient client-uploading design is crucial.

3 DESIGN

This section will explain the network design, including the C/S network architecture, working sequences of application and pivotal algorithms.

- **C/S Network Architecture:** In this TCP-based STEP protocol with a multi-threaded server, an initial 3-handshake occurs when a request is received on port 1379, allocating a dedicated thread for the client. The constructed network architecture, depicted in figure 1(a), illustrates this procedure. Upon the connection establishment, the token returned by server is indispensable for all subsequent tasks,

which can be obtained by LOGIN, an operation contained unique reserved fields such as *username* and *password*. Subsequently, the client initiates a SAVE process, delivering the *key* and *size* of the file to the server for an upload plan. On extracting the *key*, *block_size*, and *total_block*, the client transfers message blocks, each with *key* and *block_index*, to server for UPLOAD. The *MD5* of the whole file will be sent back as the last block reaches the destination to check the uploading completeness.



(a) C/S Network Architecture (b) Workflow

Figure 1: Network Design

- **Workflow of Solution:** Figure 1(b) above is an activity diagram that illustrates the entire process of the file uploading in detail. Here lists an outline for essential procedures:
  1. Upon TCP connection, the client converts the input ID to an MD5 password. Other compulsory fields, for instance, file type and operation, are packed together and sent. The server then extracts required information and generates an encoded token encompassed in the JSON data part, which will be fetched by the client soon afterwards.
  2. After obtaining the specific token, all operations, specifically SAVE and UPLOAD in this project, will undergo a Token Verification to evaluate if valid or not.
  3. Resembling the token, another vital client-defined field, key, is added and applied to acquire an upload plan in SAVE and conduct practical block-level transfer in UPLOAD. The file MD5 value calculated by the server will be compared with that in client to check uploading completeness, which symbolizes the culmination of the process.
- **Algorithm:** The kernel logic of two algorithms are shown below:

Algorithm 1 Authorization Algorithm

```
FUNCTION authorization()
    COMPUTE password using MD5(user_id)
    SEND login request with user_id, password
    RECEIVE response from server
if    error in response then
        PRINT error message
else
        EXTRACT and PRINT token
end if
    RETURN token
END FUNCTION
```

Algorithm 2 File Upload Algorithm

```
FUNCTION upload_file()
if file does not exist then
        PRINT error message
        RETURN
end if
    READ file details
    SEND file upload initiation request
    RECEIVE response and PRINT upload plan
for EACH block in file do
        SEND block
        UPDATE upload progress
end for
    VALIDATE file upload with MD5
    PRINT result
END FUNCTION
```

4 IMPLEMENTATION

- **Host Environment:** Table 1 below lists the host implementation environment in this project:

Table 1: Implementation Environment

CPU	Intel(R) Core(TM) i7-1165G7
RAM	16.0 GB
OS	Windows 11 x64
IDE	PyCharm (Python 3.9)

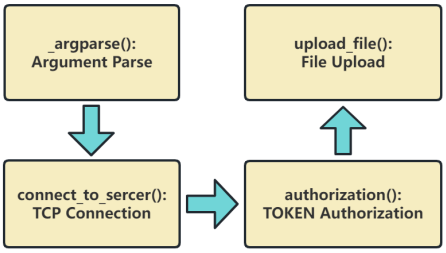


Figure 2: Program Flow

- **Programming Skills:**
    1. **Modular Programming**, one of the most significant principles that the application focuses on, where distinct functions serves diverse purposes. For example, *get_file_md5* for MD5 hash calculation. Not only the code readability and maintainability are enhanced, but also procedures for debugging are simplified due to module independence.
    2. **Abstraction** encapsulates complex operations such as data packing and token extracting into simple interfaces, which facilitates interactions between users and the system.
    3. **Procedural Programming Paradigm** calls pre-defined functions in sequence, from argument parse to TCP connection close, which moulds the application into a structured flow with working logic. Specifically, the flow chart is described in figure 2.
- **Difficulties:** Two issues regarding syntax error and module installation were identified during the

implementation. The first involved a failure in file upload to the server, attributed to a previously undetected function name bug: *data_process* to *file_process*. The second concerned the infeasibility of creating an upload bar, stemming from the lack of module *tqdm*. This was addressed by executing *sudo pip install tqdm* in the command line.

5 TESTING AND RESULTS

- **Testing Environment:** Performance tests were conducted on two identically configured virtual machines, namely Server and Client. Table 2 lists detailed environment parameters.

Table 2: Environment Configuration

CPU	RAM	OS	TYPE	Interpreter
Intel(R) Core(TM) i7-1165G7	4096MB	Ubuntu (64-bit)	Linux	Python 3.8.10

- **Measurement Techniques:**
  1. **Timer:** Transmission time, a pivotal metric for performance evaluation, relies on the imported module *time*. Specifically, *time.perf_counter()* provides a nanosecond-precision, monotonically increasing timer for precise execution duration measurement.
  2. **Progress:** While *time.localtime()* serves to represent the real-time span of the entire process, the *tqdm* module is applied to visualize the dynamic progress of the upload.
- **Testing Steps and Results:**
  1. Upon fixing all server syntax errors and running, the Server command line will display the expected message that shown in figure 3.

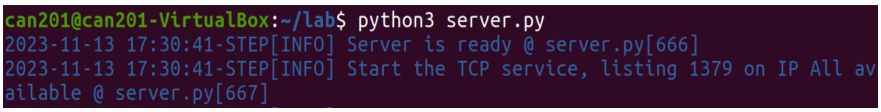
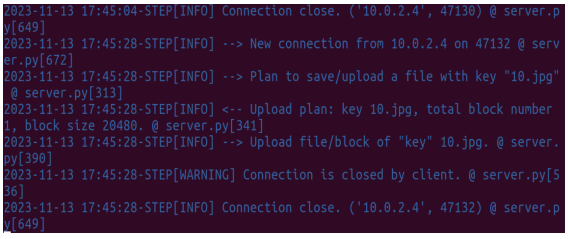
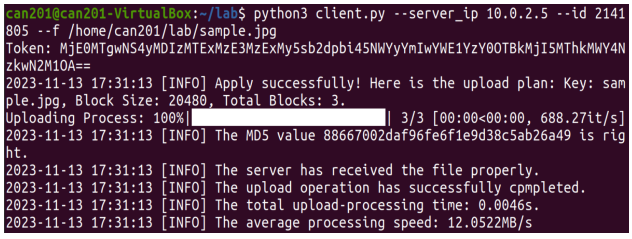


Figure 3: Server Setup

  2. After server activation, execute the *client.py* on Client, adhering to the prescribed argument format, resultant information is exhibited in following figures: figure 4(a) lists the entire activity in Server while processes in Client are displayed in figure 4(b), including upload plan, progress bar, MD5 verification and execution assessments.



(a) Server Process



(b) Client Process

Figure 4: Testing Results

3. The size of all samples involved is distributed within 1MB. According to the transfer rate, a metric derived from the execution duration, the average performance curve is depicted in figure 5. The graph sees a fluctuation, with a predominant concentration of 6 MB/s, which may be caused by abovementioned network conditions or file size.

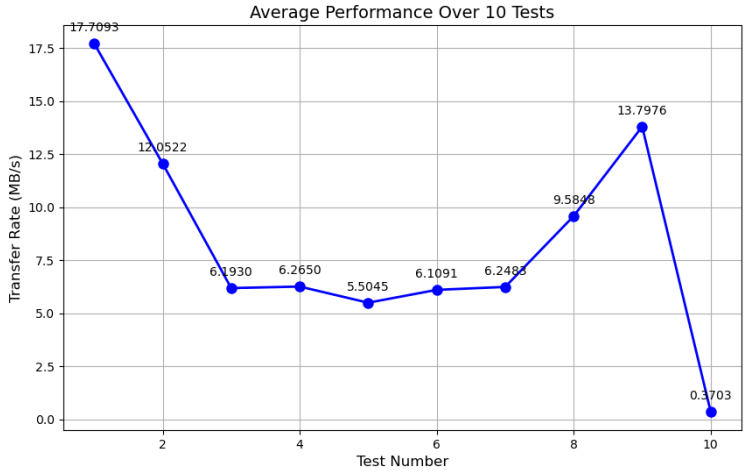


Figure 5: Average Performance

6 CONCLUSION

Overall, the project produces a reliable and efficient client-server application with a capacity of file transfer, through server code debugging, token authorization and block-level upload algorithm. Future efforts will focus on expanding the research to involve other potential processes like downloading and deleting, exploring more aspects of the STEP protocol for network transmission.

References

[1] A. Grigor'ev, E. A. Isaev, and P. Tarasov, "Transfer, storage and processing of large volumes of scientific data," 2021. 2

[2] D. C. Dogan and H. Altındiş, "Storage and communication security in cloud computing using a homomorphic encryption scheme based weil pairing," *Elektronika Ir Elektrotechnika*, vol. 26, pp. 78–83, 2020. 2

[3] E. Konidis, P. C. Kokkinos, and E. Varvarigos, "Evaluating traffic redirection mechanisms for high availability servers," *2016 IEEE Globecom Workshops (GC Wkshps)*, pp. 1–5, 2016. 2