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

INFORMATION AND COMPUTING SCIENCE

# TCP FILE TRANSFER NETWORKING PROJECT

AUTHOR: RUIJIA TIAN(ID: 2035825)
SHURAN ZHANG (ID: 2033612)
SHUCHEN YUAN (ID: 2036501)
NOVEMBER. 20, 2022

DECLARATION

I confirm that I have read and understood the University's Academic Integrity Policy.
I confirm that I have acted honestly, ethically and professionally in conduct leading to assessment for the programme of study.
I confirm that I have not copied material from another source nor committed plagiarism nor fabricated, falsified or embellished data when completing the attached piece of work. I confirm that I have not copied material from another source, nor colluded with any other student in the preparation and production of this work.

Signed: Ruijia Tian; Shuran Zhang ; Shuchen Yuan Date: November. 20, 2022

# TCP File Transfer Networking Project

*CAN201-CW-Part-I-RuijiaTian-ShuranZhang-ShuChenYuan


1st Ruijia Tian
*dept. ICS:*
*Information and Computing Science*
2035825
Ruijia.Tian20@student.xjtlu.edu.cn

2nd Shuran Zhang
*dept. ICS:*
*Information and Computing Science*
2033612
Shuran.Zhang20@student.xjtlu.edu.cn

3rd Shuchen Yuan
*dept. ICS:*
*Information and Computing Science*
*2036501*
*Shuchen.Yuan20@student.xjtlu.edu.cn*



***Abstract*— The main purpose of this experiment is to conduct simple file transfer via tcp. To maintain the security and accuracy of the transfer, the password of the user account is ciphered using the md5 encryption format, a 'token' function is constructed to ensure the accuracy of the authorisation, and a unique and specific encrypted value is returned to confirm the correctness and integrity of the file. A progress bar is also built to visualize the download process.**



***Index Terms*—TCP/IP OOT Python Transfer Socket**


## I. Introduction

### A. Background

The necessity of sending data to many devices has increased in the information era. In the current day, iCloud, outlook, google cloud, etc. can handle the vast demand for data transport. TCP/IP is the most widely used protocol for network communication. This report will attempt to complete a TCP-based file transfer protocol called STEP.

### B. Challenge and Practice Relevance

This project aims to accomplish the interaction authorization of TCP transmission, file sending, and file receiving. Our implementation includes three major steps:

Step 1. Configure the run-time environment

Step 2. debug of known server

Step 3. Create a client that can communicate with the server in an appropriate manner.

The protocol implemented has numerous potential applications. For instance, Pattaramalai [4] proposes the use of TCP frost technology on wired and wireless networks to reduce the data loss problem. Using TCP rate as the key performance setting, the advantages and disadvantages of different gel intervals are compared. The ratio of the incremental freeze time to the TCP transmission interval is calculated and it is concluded that grid congestion can more likely be reduced due to an increase in TCP throughput.

### C. Contributions

In order to make the system scalable and portable, the client code created for this course consists of the following three independent sections:

1) Sending a message request in a standard 8-byte format where the first 4 bytes is applied for the JSON data and the following 4 bytes representing the Binary data.

2) Parsing the returned message between client and server in the standard format.

3) Implementing specific operations 'login', 'save', and 'upload' based on Python. Simultaneously , a progress bar is displayed while uploading the files to make the uploading progress more intuitive.

## II. Related Work

File transfer and sharing are emerging as typical web-based applications in our daily life, e.g. Dropbox, XJTLU BOX, (J. He, 2021) [2].
Research on file transfer based on application layer has also progressively increased in recent years.

Due to the limitations of TCP/IP architecture such as data uncertainty, traffic overload, etc., a new architecture has been proposed: data network (NDN) (Tseng et. al., 2017) [7] and re-encryption schemes for NDN authentication have been made to improve its security.
(Fan et. al., 2018) [1] proposed a complete secure file transfer protocol that combines re-encryption of data to satisfy the requirement of cipher text secure transmission, addresses the potential recipient unknown problem, and saves a lot of storage cost of NDN nodes.

Lyu [6] conducted an extensive analytical study on the design and implementation issues of high-speed data transfer methods, especially the impact of application-level receives buffers and background workload on data transfer performance.
Kiran [3] investigated the feasibility of unsupervised feature extraction methods to identify anomalous patterns in networks in 2020, using three unsupervised classification methods: principal component analysis, auto encoder, and isolated forest.

Gál [8] focuses on the problem of fast communication for big data processing tasks shared among high-performance computing systems.

Yu [5] finds that NVMe-TCP is better suited for remote data reads over the network than remote data writes, and that using RAID0 in a remote network can significantly improve performance.

## III. Design

### A. C/S network architecture diagram

As depicted in the preceding C/S network architectural diagram, the client is located on the left and the server is located on the right. The solid lines indicate the queries submitted by the client to the server, whilst the dashed lines represent the crucial information returned by the server when it is running.

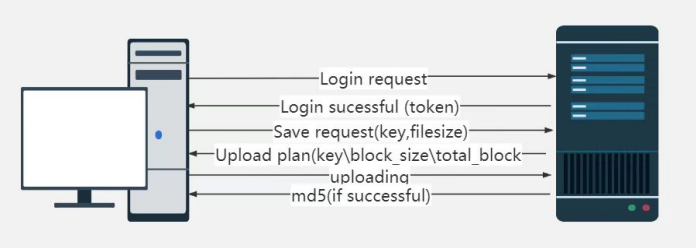


Fig. 1. C/S network architecture diagram

### B. Workflow

- *Login*
  Authorization is the first step. The client first receives command-line input, gets the information entered (user name, etc.), creates a key-value pair based on this information, and applies default information if the user doesn't enter it.
  Then the client generates a password based on the username and adds it to the key-value pairs. This includes the corresponding login information, user name and password generated in MD5 format, as well as the corresponding data type (TYPE AUTH). Everything else is left blank.
  The server receives the request, verifies the login data, and sends the client an authorization token if the data is correct.
- *Save*
  The client can request a backup once the connection has been authenticated. When a backup request is made, the client sends the relevant data to the server.The requested packet provides information on the action to be performed for the 'save' operation, as well as information of the size of the file to be uploaded as well as the key. The token obtained from the operation login is at this point also sent to the server to verify the validity of the user.
  If the user has not specified a key for the file to be uploaded, the server will generate a random key for the file. The key is unique.
  If the key already exists, the server will return an error with a specific code. Then the server returns the key corresponding to the file after the operation is executed, as well as information on the number of chunks to be split into and the size of each chunk to be uploaded.
- *Upload*
  The 'save' action is a precondition for the upload request, therefore the client can only upload after saving. For uploading, the files are transferred in blocks according to the resulting block size from the 'save'operation. The required data sent by the client to the server consists of three parts:
  part 1: the operation information requesting the 'upload' operation;
  part 2: the block size of the individual file block to be uploaded after the operation 'save';
  part 3: the the number of the initial block being uploaded.
  Namely, this is where the key and authorisation token values obtained from the 'login' and 'save' operations are delivered to the server to ensure the security and accuracy of the transfer. In this experiment, the uploaded file is transferred fully by default, so the block number of the file being transferred starts with '0' by default.

## IV. Implementation

### A. Host Environment

TABLE I
Host Environment Introduction

| CPU | Memory | Operating System |
|---|---|---|
| Intel(R) Core(TM) i5-10200H CPU @ 2.40GHz 2.40 GHz | 64-bit operating system x64 based processor | RAM 16.0 GB |

This project employs object-oriented programming to implement login, save, and upload in distinct modules. Also included is a way for displaying the progress bar. The major functions listed above will be called collectively to achieve the required functions and receive matching functions and data in different cases.

### B. Program flow charts*

*Note: The picture was put on the next page.

### C. Function Implementation

- *Login*
  The login starts first. The client-side method ' argparse' is able to receives a fixed-format message from the command prompt and determines whether the user has entered a username. After the previous operation, direction of the message transferring (client to server) is set by using the command 'json data [FIELD DIRECTION] = DIR REQUEST', meaning the direction is from client to server, and the value of the username received by the client will be assigned to the global variable 'username' for storage, in which way enable the username to be recognized during the whole process.
  Then, according to the task requirements, the global variable "username" is encrypted with md5 to make the password. Next, pack the information up to make it ready to be delivered. Here a written function called 'make request packet' is for assigning the specified input parameters to key-value pairs of JSON data and packing the input parameters into a specific 8-byte message format that can be received by the server.Here, in order to

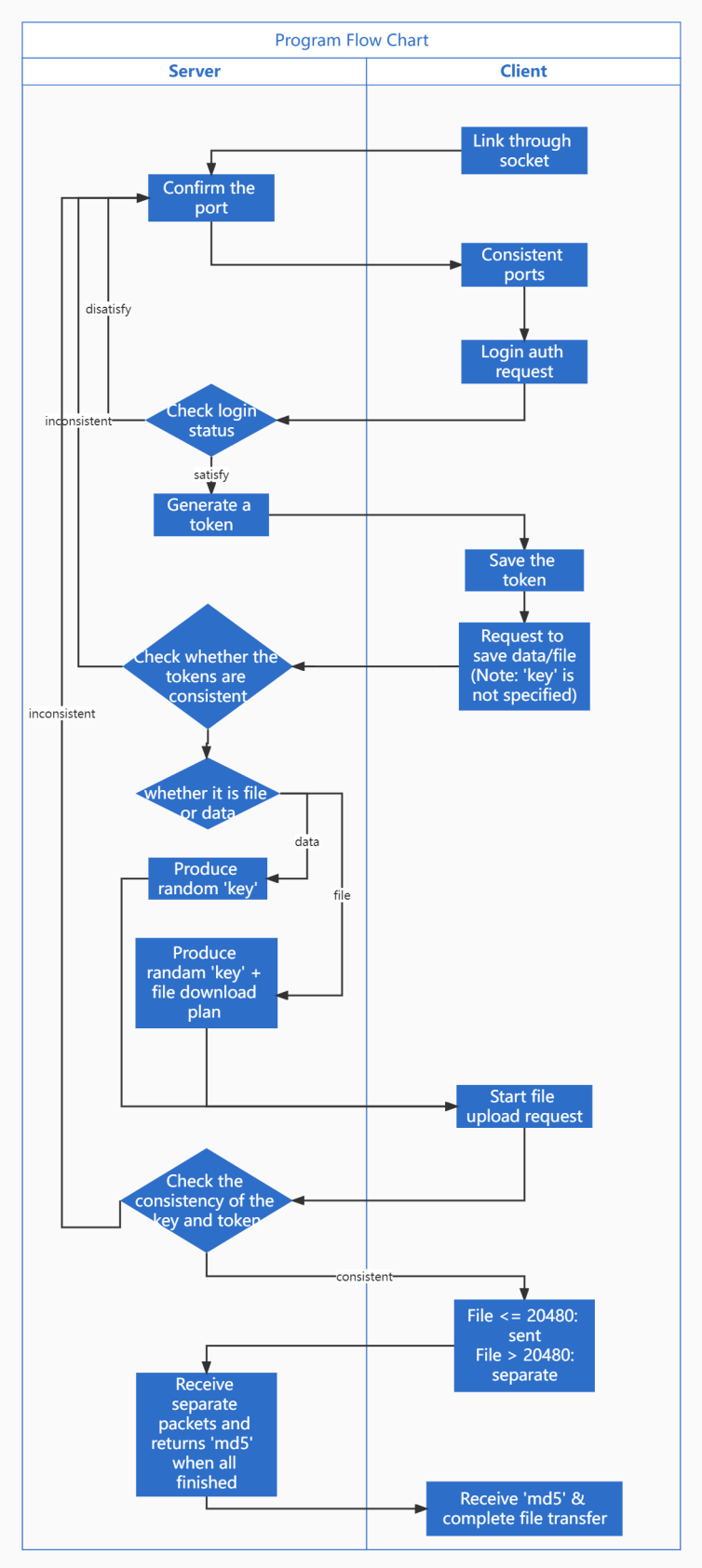


Fig. 2. program flow charts

send the request to login, in this function, the parameter value of 'operation' to 'OP-LOGIN', the parameter value of 'type' to 'TYPE-AUTH', and assign username and password values to the corresponding parameters.
Then the login information is put into a package, and the 'send' method in the socket library is used to send the whole package to the server. If the server successfully catches the request, it will generate a unique and random token, and return it to the client via the same protocol. Client use 'get tcp packet'function' is used to break up and store the server response messages, including the token message. After receiving the right token, the user successfully logs in and is authorized.

- *Save*
Before the document can be uploaded, it must be saved using the key and size of the file to send a save-request, as specified in the design. Since the initial value of key to "None" is set in the ' argparse' method, if the user does not input "key" in the command prompt, key will always be "None." Based on this, the server can determine whether or not to generate a random key based on whether the user provides the value of 'key'.
Similarly, 'get-tcp-packets' function is used to receive the 'upload plan' sent by the server after getting the upload-request. The plan includes the values of returned 'keys' and total upload blocks numbers as well as the size of each block which is assigned to the global variables 'key,' 'TOTAL-BLOCK,' and 'BLOCK-SIZE' for further use in case.
So that they can be utilized later in the upload process. After client receives a correct upload plan, the save function is implemented which means the file is ready to be uploaded.
- *Upload*
Likewise, on the premise of saving the file and obtaining the upload plan, the program can upload the file. After getting the save request and the values of the parameters 'key', 'block-size' and 'total-size', a for-loop is applied to iterate data of the uploading file by blocks where the 'open (path, 'rb')' function is used to read the contents of the uploaded file. The for loop used here allows the program to iterate over all the blocks that need to be uploaded.
In the loop, the 'seek()' method is used to move the pointer in each traversal by an amount equal to the size of the uploaded file block, as a consequence of which each file block can be uploaded in its entirety.
As following, the 'perf counter()' method in Python's time package is used to record the time of the beginning of uploading after the preceding step is complete.Similarly, the required 'key,' 'block-index,' 'token,' and 'bin-data' information is then packed up to the server using the 'make request packet' and the 'send' method as previous to deliver the file blocks to the server. Note that the value of block index is the iterator 'i', and the value of bin data is the contents of the file that

was read using 'open(path, 'rb')'. These make sure the file is uploaded by required block index.Finally, utilize the 'per-counter()' again to monitor upload progress and duration by calculating the difference between the two times. Print each upload's progress bar and the acquired JSON information. If the file is upload completely and correctly, the client will obtain a reply from the server where a matched MD5 value for the upload file is given. It means that the uploading progress is finished successfully.

The progress bar shows that the file has been uploaded a hundred percent of the way. The client prints the md5 value, which shows that the server has received the entire file. At this point, the process of uploading is finished.

## V. Testing and Result

### A. Test Host Environment

TABLE II
Test Host Environment Introduction

| CPU | Memory | Operating System |
|---|---|---|
| Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz 1.50 GHz | 64-bit operating system x64 based processor | RAM 16.0 GB |

### B. Testing Steps

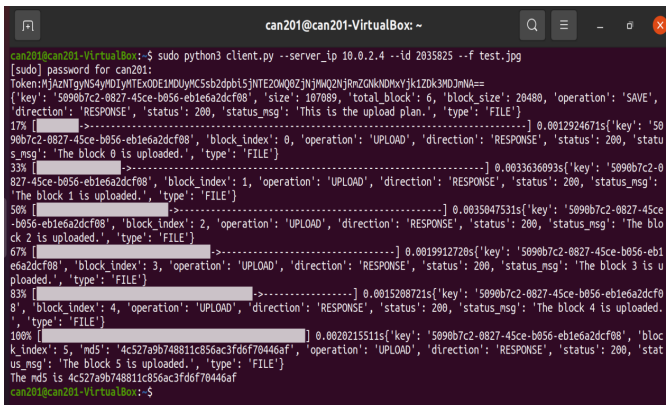


Fig. 3. Run screenshots

Both vertical and horizontal measurements were used to compare average performance. The data obtained from ten tests with four files of different sizes resulted in the following graph. It is apparent that each file took the longest time to transfer the first time, with subsequent times decreasing and transfer rates levelling off.

After searching the literature [4], it is speculated that this is due to the fact that the initial connection was set up as a long tcp link, and that the connection between the two was not actively closed after the first request was made, then the subsequent operations would continue to use the connection. In short, the server kept a semi-open connection, so subsequent times were reduced and the first time took a longer time.

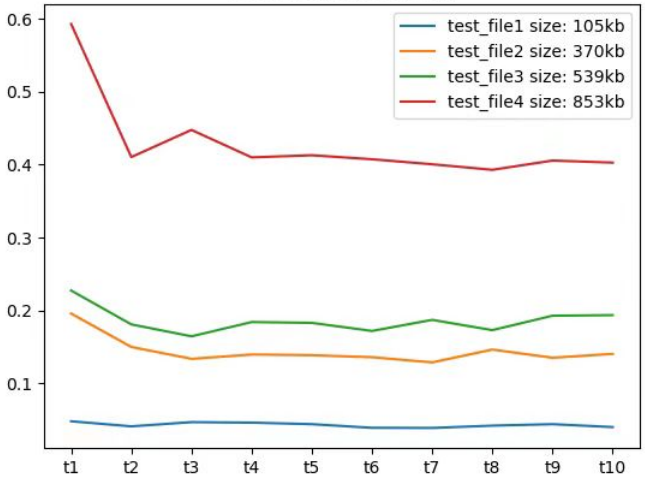


Fig. 4. The performance result

## VI. Conclusions

Our group defined an application layer protocol for TCP-based file transfer, designed a file transfer system with login authorization, and implemented the system using Python and pycharm. This system has passed performance tests and can transfer packetized files or data.


## Acknowledgment

All three of us have made a serious effort to contribute the same percentage to this project. Meaning, Ruijia Tian (2035825) contributed 1/3, Shuran Zhang (2033612) contributed 1/3, and Shuchen Yuan (2036501) contributed 1/3 to this project.


## References


[1] Chun-I Fan; I-Te Chen; Chen-Kai Cheng; Jheng-Jia Huang; Wen-Tsuen Chen; "FTP-NDN: File Transfer Protocol Based on Re-Encryption for Named Data Network Supporting Nondesignated Receivers", IEEE SYSTEMS JOURNAL, 2018. (IF: 3)

[2] J. He, "A Self-design Protocol of Transport Layer to Implement File-Sharing Box," 2021 2nd International Conference on Big Data and Artificial Intelligence and Software Engineering (ICBASE), 2021, pp. 705-709, doi: 10.1109/ICBASE53849.2021.00138.

[3] Mariam Kiran; Cong Wang; George Papadimitriou; Anirban Mandal; Ewa Deelman; "Detecting Anomalous Packets in Network Transfers: Investigations Using PCA, Autoencoder and Isolation Forest in TCP", MACHINE LEARNING, 2020.

[4] S. Intarapanich and S. Pattaramalai, "Increasing TCP performances on wired and wireless networks by using TCP freeze," 2016 International Computer Science and Engineering Conference (ICSEC), 2016, pp. 1-6, doi: 10.1109/ICSEC.2016.7859904.

[5] Se-young Yu; "Performance Evaluation of NVMe-over-TCP Using Journaling File Systems in International WAN", ELECTRONICS, 2021.

[6] Xukang Lyu; Chase Qishi Wu; "End-System Aware Large File Transfer Solution for Rich Media Applications Over 5G Mobile Networks", 2020.

[7] Yi-Fan Tseng; Chun-I Fan; Yan-Fu Cho ”An authenticated re-encryption scheme for secure file transfer in named data networks”, doi: 10.1002/dac.3571

[8] Zoltán Gál; Gergely Kocsis; Tibor Tajti; Róbert Tornai; ”Performance Evaluation of Massively Parallel and High Speed Connectionless Vs. Connection Oriented Communication Sessions”, ADV. ENG. SOFTW., 2021.