CAN201 CW1 Development Report

Bigstool

School of Advanced Technology

Submitted to

Xi'an Jiaotong-Liverpool University

Abstract

In the context of personal cloud storage services increasingly gaining popularity, Coursework 1 specifies a file-sharing system with the characteristic of Large Efficient Flexible and Trusty (LEFT). This report summarizes the author's approach to the coursework regarding methodology, implementation, and testing. Covering the component and protocol design, system sequence diagram, implementation steps, difficulties, and testing details. The coursework has promoted extensive use of the contents covered in the module. Despite the immaturity of the produced system compared to products on the market, the system can serve as the foundation of future learnings and projects.

1. Introduction

1.1 Background

Over recent years, rapid developments have taken place in the field of cloud-based services, offering various business solutions [1]. Personal cloud storage services, which is intended to simplify the process of synchronizing files over several devices [2], in particular, has gained prominent popularity as well as momentum amongst both enterprises and internet users [2], with long-established enterprises such as Dropbox which have attained tens of millions of users [1], and internet giants including Google and Microsoft entering the market [2]. In the context of the fast-evolving personal cloud storage services, Coursework 1 specifies a simplified file-sharing system that provides the functionality with the characteristic of Large Efficient Flexible and Trusty (LEFT).

1.2 Project Requirement

According to the task sheet of Coursework 1 [3], the project is required to implement a Large Efficient Flexible and Trusty (LEFT) file-sharing system. To be more specific, given the IP addresses of the peers, the system should provide the function of file synchronization among peers, which supports folders, files larger than 1GB, compression of files, partially synchronize changes made to file (around 0.1%

continuous part at the beginning of the file, and encryption. The system is also expected to resume downloads when being killed and restarted.

2. Methodology

2.1 Component Design

A total of 6 components were identified and designed, namely Connection Hub, File Center, Download Manager, Compression Station, Encryption Bureau, and Main.

Connection Hub acts like a post office that sends and receives messages to/from peers. Messages produced by the components are first sent to the Connection Hub, then sent to the peers' Connection Hub, then sent to destination components.

File Center keeps records of local files, informs other peers of new files or modified files. File Center also provides the read file function, allowing specified blocks of a file to be read and sent to peers.

Download Manager, provides the download management functions, which mainly include checking file information sent from peers, schedule and manage (new or resumed) downloads, and partial updates.

Compression Station utilizes the gzip module to provide the compression functions to reduce the size of data sent to the peers, hence reduces synchronization time.

Encryption Bureau utilizes the pycryptodome module to provides the encrypt and decrypt functions with AES256, making data security during transmission possible.

Main is the entrance of the system, which is the program run in the terminal when starting the system. Main initializes and starts other components.

2.2 Proposed Protocols

The protocol for the LEFT system was designed to support messages of variable length and type. Therefore, the basic format of the protocol was designed as follows:

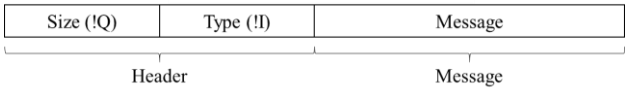

Fig 1: Protocol Format

The header consists of two parts: the message size, and the message type. Message size indicates the length of the message, while the message type indicates the type of the message. The header is appended to the beginning of the message.

A total of 6 types of messages were proposed for the LEFT system: Encryption (0), File Dict (1), File Modified (2), File Added (4), Block Request (5), Block (6), which in order states the peer's encryption preference, informs peers of all currently held file information, informs peers of a modified file, informs peers of an added file, requests a block from a peer, and sends a block to a peer. The format of each type of message is as follows:

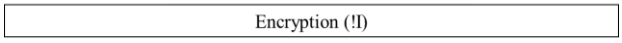

Fig 2: The Encryption Message

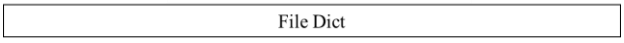

Fig 3: The File Dict Message

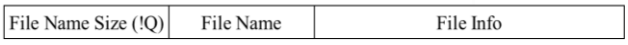

Fig 4: The File Modified / File Added Message

Fig 5: The Block Request Message

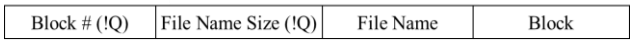

Fig 6: The Block Message

3. Implementation

3.1 Overview

Fig 7 and fig 8 below shows the simplified sequence diagram of the upload and download workflow. It should be noticed that the initialization by Main and encryption by Encryption Bureau is omitted. When the system is started, Main initializes other components. If encryption is set to yes, every outbox message will be first encrypted and every inbox message will be first decrypted, with an exception of message type 0: Encryption, which is always transmitted without encryption.

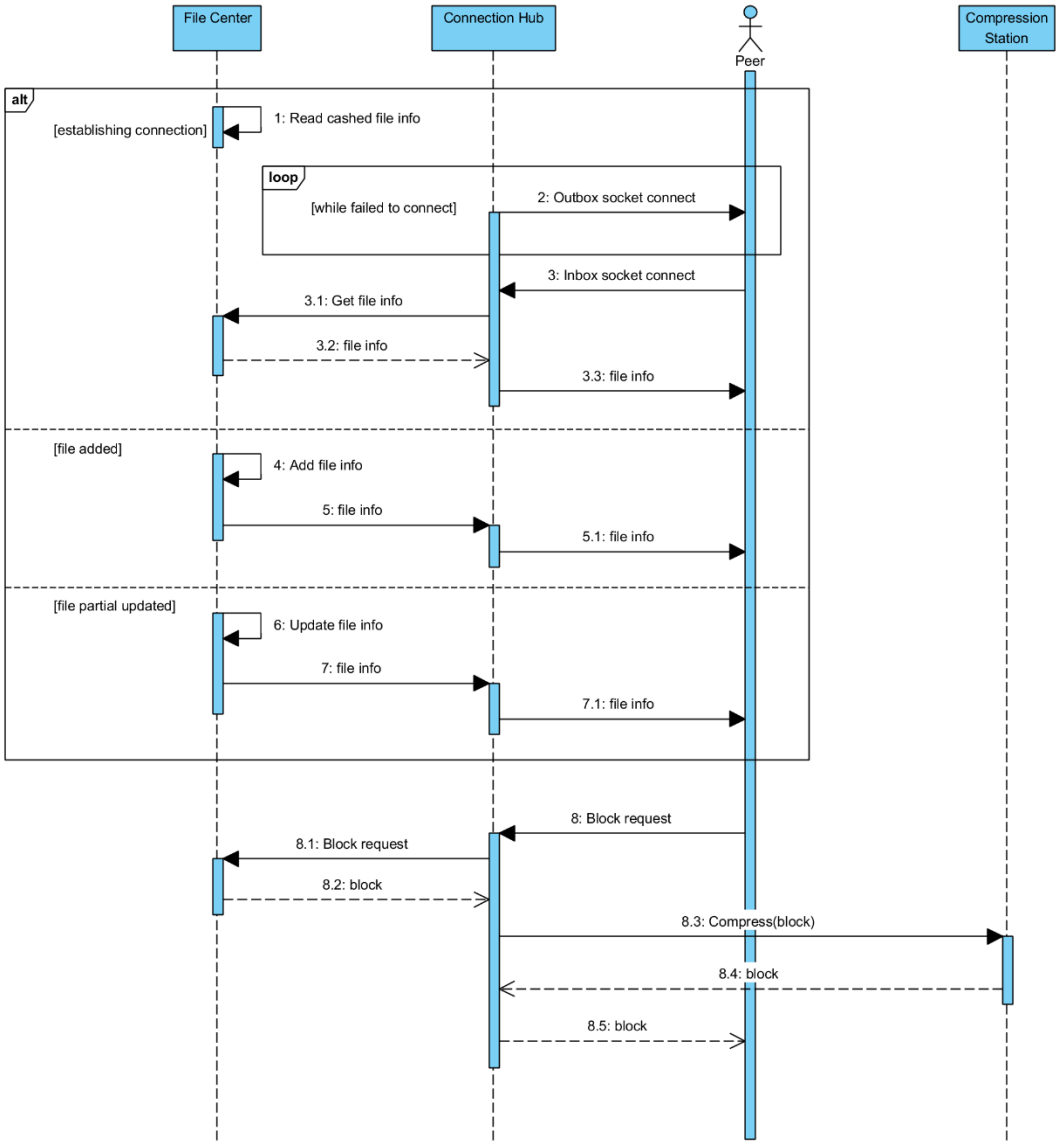

Fig 7: Upload Overview

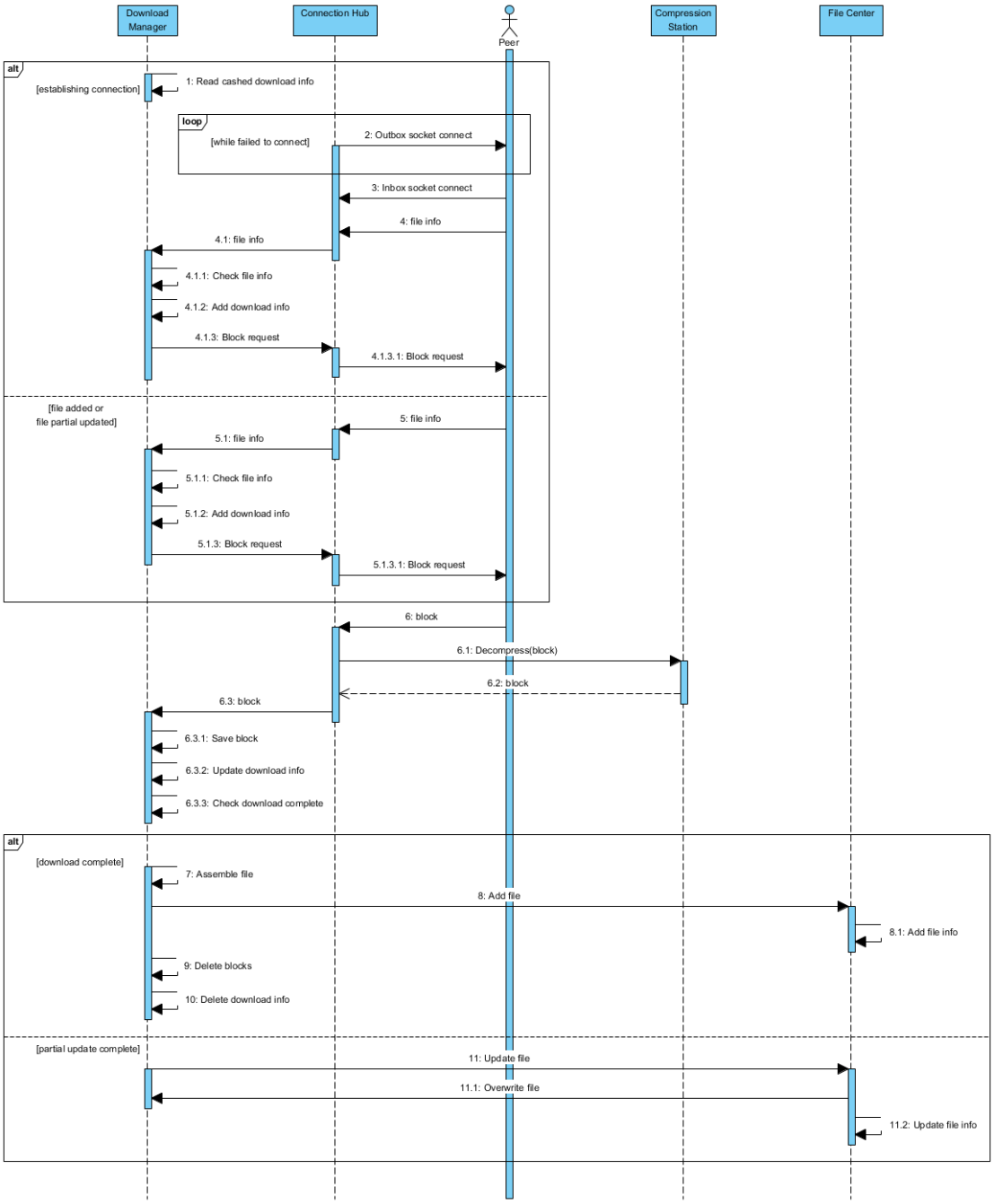

Fig 8: Download Overview

3.2 Implementation Steps

The first feature implemented was the local file read and management feature, which later supported the development of File Center. MD5 checksum was originally used for file modify detection, which was quite time-consuming. Since the file modify test only modifies around 0.1% of the file at the beginning, a workaround was later adopted using only the file modified time acquired using the os.path.getmtime()

function.

The second component being implemented was the Connection Hub. A two-way TCP connection scheme was implemented: each system runs an outbox thread that constantly attempts to connect to peers before the connection is accepted, and an inbox listener that accepts inbox connections from peers' outbox thread, then passes the socket to a new inbox thread. A chat application was developed based on Connection Hub to test the component.

For the two important complements of the Connection Hub, Compression Station was the third component implemented. However, the implementation of the encryption functions was delayed, with some code portion reserved for encryption in the Connection Hub. Encryption Bureau was the last component implemented.

With the experience from developing the previous components, Download Manager was developed after Compression Station without much difficulty. Followed by the implementation of Main, which integrated the components developed into a working system.

3.3 Encountered Difficulties

Various difficulties were encountered during the implementation of the system. Including sending messages between threads, slicing received TCP byte stream into packages (headers and messages), implementing compression and encryption, etc. The Python document and Communities including Stack Overflow and CSDN have been a great support. A brief credit list can be found at https://github.com/Bigstool/CAN201_CW1/blob/main/Credits.

4. Testing and Results

The system was tested using three computers. One of which was running Windows, and the other two running Linux. The three computers are in the same internal

network, and the test has followed steps 1–15 of the code testing steps specified in the task sheet.

The results suggested that the system is able to correctly perform the tasks required by each step. However, since the system will be tested on virtual machines instead of separate computers when the coursework is being marked, the actual download/upload speed may be faster due to the utilization of the virtual network adapter.

Compression and encryption, which is specified in steps 16 and 17 in the code testing steps, is also tested individually. The time consumption for a computer to download a file with a size of 207MB in three different configurations (all disabled, encryption-only, compression-only) is shown in fig 9 below:

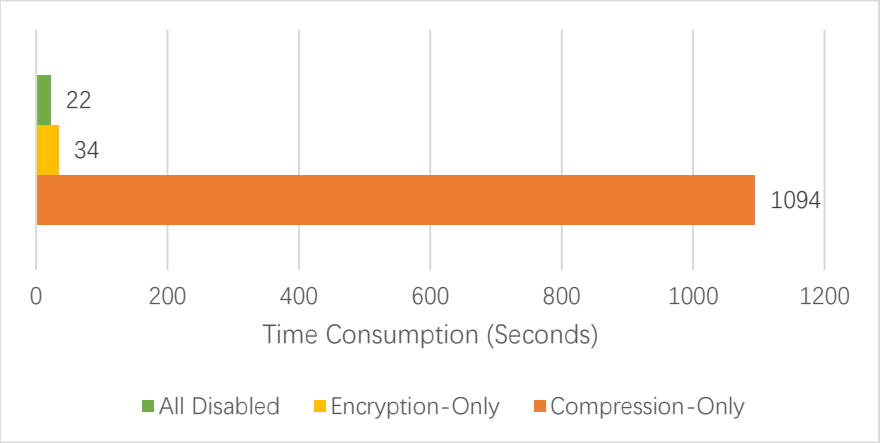

Fig 9: Time Consumption

Surprisingly, downloading the file with compression consumed different order of magnitude than the other two configurations. This is arguably an ironic result as the introduction of compression was originally intended to increase download speed. One possible reason is that the gzip module may be implemented in pure Python, hence has low efficiency. Therefore, compression may be disabled when the coursework is submitted.

5. Conclusion

A LEFT system was implemented through the process of component and protocol design, implementation, and testing. The implementation of this system serves the purpose of helping the students to better understand the contents and mechanisms of computer networks, especially the application and transport layer. The system alone may be regarded as the prototype of a file-sharing system, which may be of referencing value to a file-sharing system product.

Unfortunately, while the current system may meet the requirement of the coursework as a school project, it is arguably immature with various limitations lying within. For instance, it does not support file/folder deletion or empty folders. It cannot partially update files which are modified in arbitrary positions, or handle modified file where the size of which is changed. The shortcomings mentioned above may be addressed as a part of the future plan.

References

[1] I. Drago, M. Mellia, M. M. Munafò, A. Sperotto, R. Sadre, and A. Pras, "Inside dropbox: Understanding personal cloud storage services," *Proc. ACM SIGCOMM Internet Meas. Conf. IMC*, pp. 481–494, 2012, doi: 10.1145/2398776.2398827.

[2] E. Bocchi, I. Drago, and M. Mellia, "Personal cloud storage: Usage, performance and impact of terminals," *2015 IEEE 4th Int. Conf. Cloud Networking, CloudNet 2015*, pp. 106–111, 2015, doi: 10.1109/CloudNet.2015.7335291.

[3] F. Cheng, "Large Efficient Flexible and Trusty (LEFT) Files Sharing," Suzhou, 2020. [Online]. Available: https://learningmall.xjtlu.edu.cn/pluginfile.php/52734/mod_assign/introattachment/0/CW1.pdf.