COMP90015 Distributed Systems Project 2 Report
BitBox

Lecturer: Dr. Aaron Harwood

United Ace Members

Name	ID	Login Name	E-mail
Wanmin Chen	941065	WANMINC	wanminc@student.unimelb.edu.au
Junjie Huang	1016904	JUNJHUANG	junjhuang@student.unimelb.edu.au
Tong Mu	1004452	MTM	mtm@student.unimelb.edu.au

School of Computing and Information Systems
University of Melbourne
Australia

June 2019

1 Introduction

In this project 2, we aim to implement a Peer-to-Peer (P2P) file distributed system called BitBox. Basic protocols have done in Project 1, and two main tasks are included in this project: One is that to build a BitBox client which can securely communicate with the BitBox Server by using TCP with public/private key (ssh-keygen) and secret key (AES128), and the other is to implement BitBox server to use UDP, in lieu of TCP between other peers. The project structure is presented as Figure 1.

Two topics are discussed in this report: One is that how public/private key technique could be used to provide a file permission scheme in BitBox, and the other is the comparison between UDP implementation and TCP approach.

2 File permission Scheme

Regarding the issue of handling read and write permissions, the basic idea of our design is that we identify three classes, which are owner, group and others. Owner is the creator of file/directory, which is initially defined as having the permissions of read and write to this file/directory. A peer could get the permission of read and write for a file/directory after joining the group with owner's identification. Members in others only have the permission of read, which is the default situation. In other words, if a new peer joins into this P2P network, it has the right to read files in the share directory but does not has the right to write the files. If the new peer is willing to write this file, it must send a request to the owner to ask for the write permission.

Figure 2 shows the interaction diagram of our design about how to use public/private key technique in BitBox between the communication with each other. The situation in Figure 2 shows the process where the Peer1 wants to get the writing permission of F1.txt from Peer2, which is the owner of this file.

Peer1 will send Peer2 a request including a secrete message, which is encrypting by using the identity of Peer1 and the public key of Peer2. After receiving the request, Peer2 will decrypt the secrete message by using the private key of Peer2 and get the identity of Peer1. If Peer2 (the owner) agrees to give the write permission to Peer1, he/she will add Peer1's identity into the Identity List for the file (F1.txt), and then broadcast the new identity list to every peer. Finally, send a response of the result of this permission request to Peer1. Therefore, after receiving the successful response, when Peer1 is willing to write F1.txt, the system will check whether its identity is in the identity list of this file, and finally find its identity, so the system will give the write permission to Peer1. It is worth to say that changing identity list is modifying an attribute of the file/directory, therefore, it will be directly monitored by FileSystemManager, which is already implemented in Project1.

If a new peer Peer3 adds into the network, the system will firstly generate a pair of public key and private key. Private key is stored by itself, and its public key and identity will be broadcasted to every peer by the system. Afterwards, the system

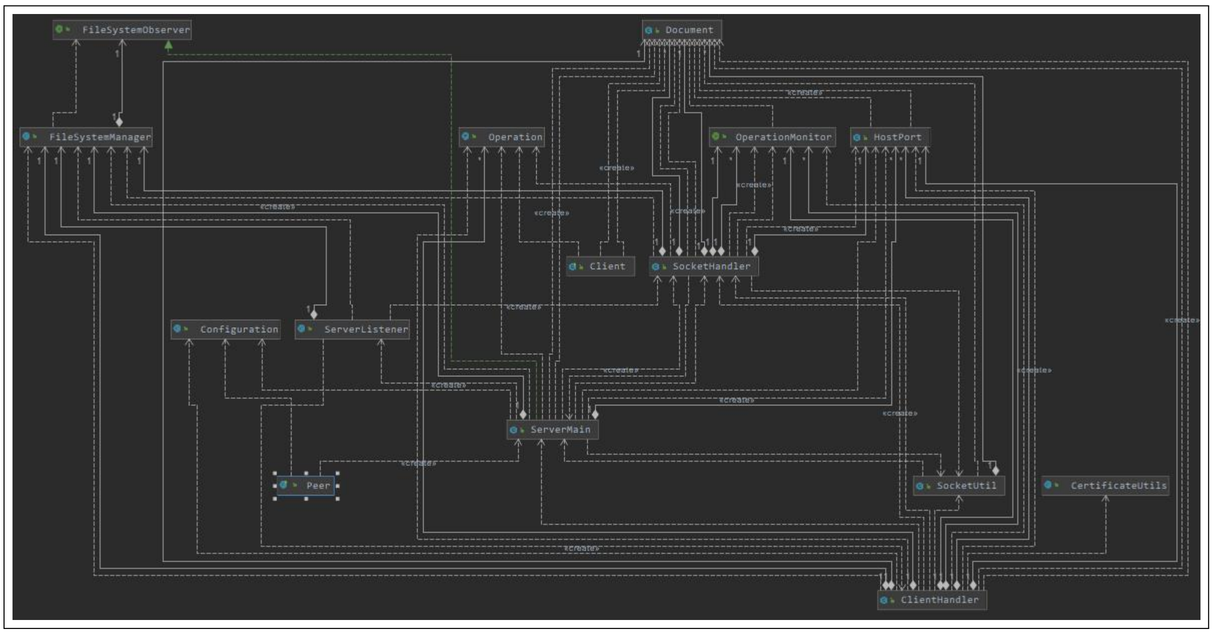

Figure 1: Project2 structure

will add Peer3 into others class with the permission of read, and then synchronize all of the files in the share directory. Therefore, Peer3 only has the permission of read at the beginning, and if he/she is willing to modify some files, the steps will be followed as Figure 2. However, the time of this permission is short, in other words, this file can be modified repeatedly in a short time (can be 1 hour or 6 hours) decided by the owner (Peer2), but after this period of time, if peer1 wants to write this file again, then it must re-apply for the permission. But if the system detects that Peer1 is still modifying the file when the time is out, the system will automatically extend the time until Peer1 completes the modification. The benefit of this mechanism is that it can improve security of the P2P network. Meanwhile, a file won't be allowed to modify by several users at the same time, which means other users who also have the write permission will be locked until the writer is finished.

The protocols which will be implemented to achieve this goal are as follow:

1. RequestWritePermission: a protocol to send owner a request to get file write permission using the secrete message, which is encrypted by user's identity and owner's public key.

2. ResponseWritePermission: a protocol to use owner's private key to decrypt the secrete message in order to get the information of user's identity and send response to the sender.

3. AddIntoGroup: a protocol to add the identity of user into the identity list. For example, the user will join into group class with permission of read and write after owner's identification successfully.

4. RemoveFromGroup: a protocol to remove user's identity. For example, the system will remove user's identity automatically when the user is off-line.

5. SetPermission: a protocol to set permission of a file/directory for three classes, which are owner, group and others. The creator of a file/directory is assigned to owner class directly; other peers will be assigned to others group automatically at the beginning; group class will be assigned after owner's identification.

6. GenerateKeys: a protocol to generate peer's public key and private key as a pair. For example, when a new peer joins into the network, this protocol will be used to generate its public key and private key.

7. BroadcastPublicKey: a protocol to broadcast peer's identity and its public key to every peer.

2.1 Limitation

First of all, from the perspective of security, the write permission requests between the peer and peer are encrypted by the public/private key technique, if one of the peer's private key is leaked and the peer does not know it, it is easy to be third-party used, posing as the identity of this peer, thus this situation will bring a potential crisis to the entire P2P network. Therefore, this protocol is still somewhat inadequate in identifying the identity of the requester. Secondly, in order to avoid unrestricted modification of the same file, each authorization is limited to modify the document during a period of time, which is decided by the owner. But if there is a non-creator who needs to modify the document frequently, then this protection mechanism of the protocol limits the flexibility of the non-creator to modify the document. In addition, since the written authorization request is sent to the owner of the file, the file only can be modified after getting the permission form the owner. If the owner does not accept the request or the owner's server is down for some reason, then the next step can only be performed after the owner's state is restored. This also limits the flexibility of the operation. Furthermore, since each file requires authorized operation, in the case of a small probability, if there is a batch file processing request from

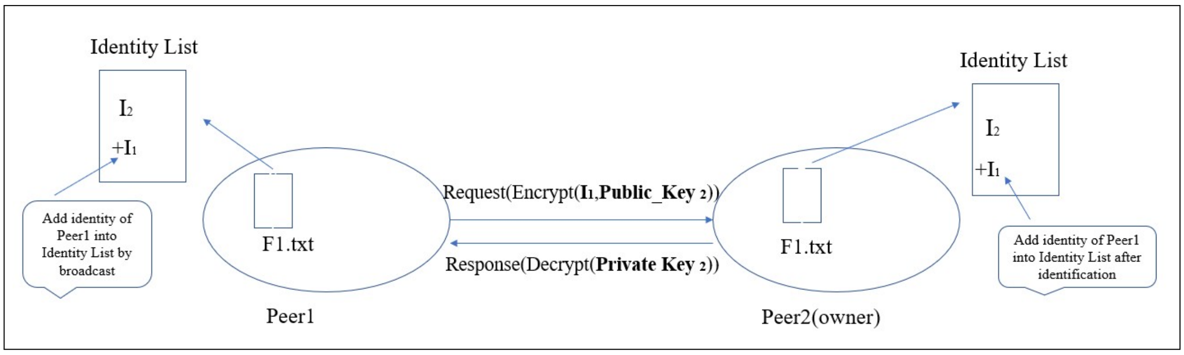
Identity List
I2
+I1
Add identity of Peer1 into Identity List by broadcast
F1.txt
Request(Encrypt(I1,Public_Key 2))
Response(Decrypt(Private Key 2))
F1.txt
Peer1
Peer2(owner)
Identity List
I2
+I1
Add identity of Peer1 into Identity List after identification

Figure 2: Interaction diagram

a peer, the peer needs to send many requests and the system has to perform corresponding permission processing for each file, which will cause the authorization operation to become Cumbersome.

3 Strength of UDP Use

After transferring files by using the UDP protocol, it is obvious that the transmission speed of files, especially large files, is improved compared to the previous TCP protocol. This is related to the UDP transport mechanism, because it is not necessary to perform three-way handshake, port acknowledgement and other mechanisms like the TCP protocol. It is a stateless transport protocol, so the file transfer speed is faster than the TCP protocol. In addition, since the UDP protocol does not have acknowledgment mechanism and a handshake mechanism like the TCP protocol, attacks like DOS and DDOS can be avoided, and system security is slightly higher than when using TCP transmission. However, it is precisely because of these characteristics of UDP that it does not retransmit the packet due to packet loss during the file transmission process, which leads to when the network quality is not good or the frequency of sending packets is too fast. It is easy to lose packets. For example, the team used the TCP protocol to test the connection between the peer and the peer in the project 1. When the handshake was successful, the peers communicated with each other and the file was transmitted without packet loss, but when the transmission protocol was changed to UDP, packet loss often occurs. In the initial stage of program development, when the packet loss situation was not resolved, the packet loss rate reached 30 percent. Later, in order to solve this problem, the group proposed using a Hash map to detect whether packets were lost. The system will check the time of each "document.getLong" in "System. nanoTime()-entry.getValue()", if the time is less than 11 seconds, the system will treat it as packet loss happened and the system will re-transmit it. In addition, some peer file paths are in Chinese, so in this program, the system uses the parameter "actual length of the encoded byte" to address this problem when sending and receiving. Furthermore, when connected to other peers, if it does not send a message to the other peer for a while, then the other peer will disconnect this connection. In response to this problem, the team's idea is to optimize the system by adding heartbeat detection, that is, to periodically send a small package to the other peer to make sure that the system is always in the connected queue.