**FILE TRANSFER PROJECT DOCUMENTATION**

By

BAO NGUYEN – BCN140030

MOFAN LI – MXL162930

CS 6349.001

1. **Project Description and Solution**

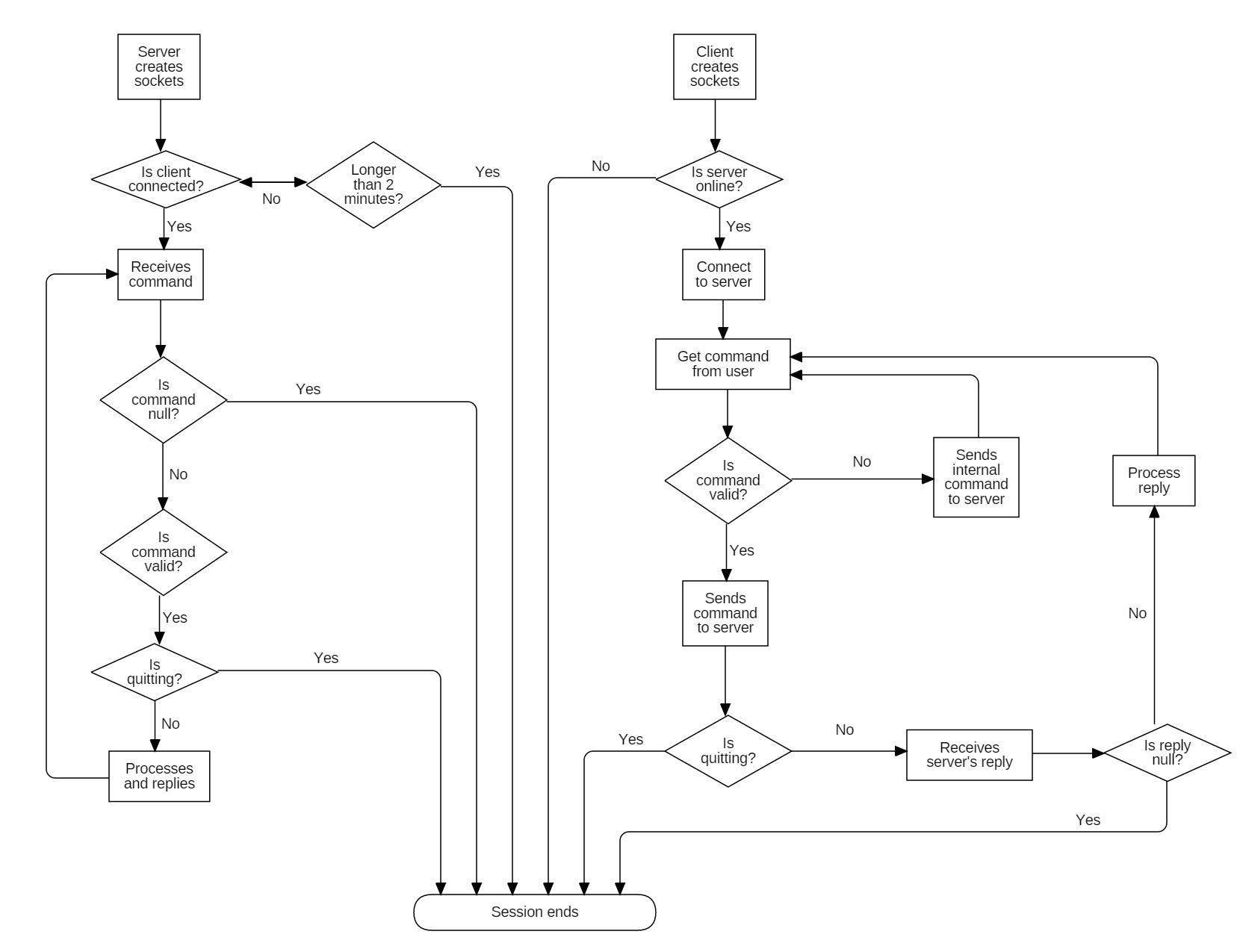
The problem is to create a file transfer system that utilizes Socket Programming, Connection Management, Reliable Communication, and security protocol that utilizes SHA-1. It allows 2 individuals (1 server and 1 client) on different computers to communicate via network. This project supports Windows, Mac, and Linux.

We use Java to build this program. We create 2 different projects/modules, a *Client.java* and a *Server.java*, both lie within the *FileTransferApplication* project. The server sets up the sockets and wait for connection. The client sets up its socket and connect to the server. This program also records the start and end time of the session.

There are several commands that allows the client to communicate to the server. The server does not initiate any message. It waits for the commands from the client and sends responses. The client can request server to view all the files the server contains, to download files from the server, or to upload files from local host to store in the server. All messages transported over the network shall be secured with respect to Authentication, Confidentiality, and Integrity.

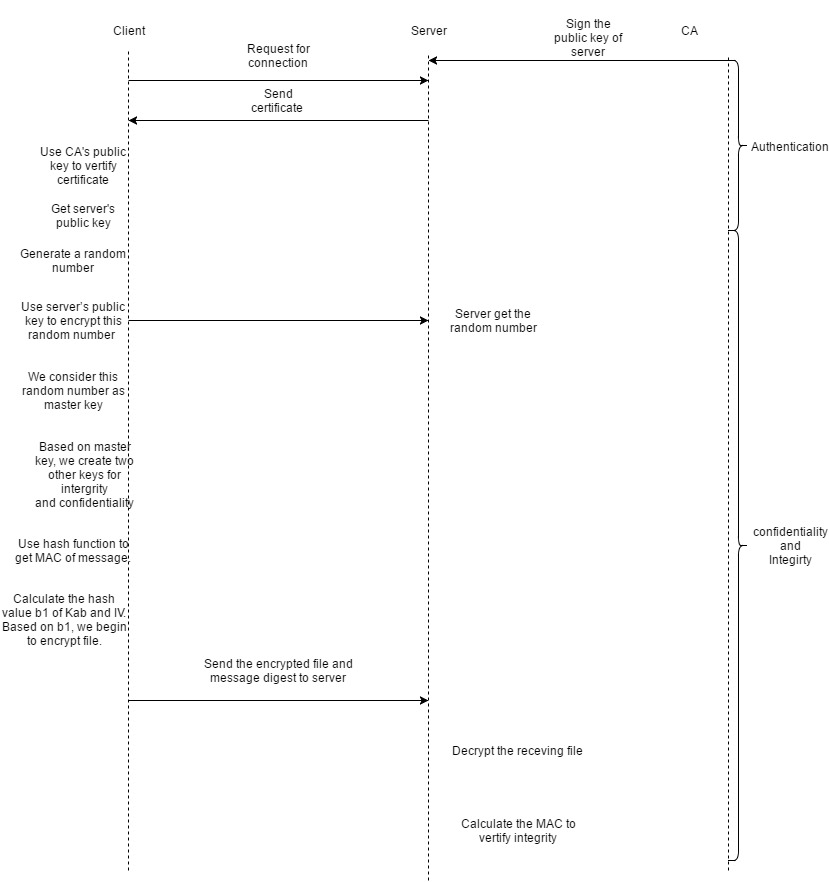
Since Java forces to close the stream in order to finish sending a file as byte stream, we has to close the socket after each transaction (send message and receive message together is consider a transaction) and reopen new socket (same server’s IP and port with the previous socket) for another transaction. Therefore, there are multiple sessions between client and server during run time. The first session, called S0, performs a major authentication (key exchanges, server remembers this client, etc.) and comes up with a session key K0. In the follow-up sessions, the server first will make sure it still talks to the same client by validating client’s IP address and ID. This is the only authentication that the server performs on the client. [Describe how client authenticate server]

There are several occurrences when the program ends (the flow of the program is provided in the next section of this report):

1. If the server waits too long without any connection from the client, the program ends at that point.
2. If the client tries to establish a connection without an active server (on the other word, the client starts before the server does), the program ends immediately.
3. After the connection is established, they can continue communicating (the client starts first) until the client says “quit”. When this event occurs, all sockets are closed and the program ends.
4. During the session, if either the server or the client goes offline for any reason (indicating by a null message received by the other party), the one who remains online displays an error message to the screen to indicate the event and the session ends safely (without crash).
5. **Implementation**
   1. *Abstract Flow Chart*

*Basic flow of program. Authentication mechanism will be added as separated diagram.*

* 1. *Security Protocol*

**

1. **Code Executions**

Before running the program, user has to make sure the client and server are on the same network. Each of them can be on different machine, but both of them have to connect to the same network. If they are on different networks, the server machine has to setup port forwarding before it can operate with a remote client.

There are two ways to run the system: via IDE or via Command Line environment.

1. To run program via IDE (this was developed via IntelliJ IDEA and Eclipse), first open project FileTransferApplication. Then, individually run Server module and Client module. From this step just simply follow the prompts.
2. To run program in the command line environment, first navigate to where the src folder of the Server module and the Client module located on your local machine (should be subfolders of FileTransferApplication folder.) Run command “javac Server.java” for Server and “javac Client.java” for Client to compile the code. Then run “java Server” to start the Server and “java Client” to run the Client. After this, run as program instructs.
3. **Detail About Security Protocol**
   1. *Authentication*

Server has a certificate which is its public key signed by CA’s private key. The client has CA’s public key. When server receiving the connection request from client, server will send the certificate to client. Client uses CA’s public key to verify the certificate and get server’s public key. After authentication, client generates a random number. We consider this random number as a master key.

Encrypt master key with server’s public key and send it to server.

* 1. *Confidentiality*

In this project, we use Sha1 to encrypt our files which is generating and using one-time pass to XOR with data.

b1 =Sha1(Kab|IV) c1=p1⊕b1

b2 =Sha1(Kab|c1) c2=p2⊕b2

… …

bi =Sha1(Kab|ci-1) ci=pi⊕bi

* 1. *Integrity*

We also use Sha1 to calculate the MAC(Message authentication codes).

MAC =SHA1(Kab|m)

1. **Threat Models**
   1. *Eavesdrop*

Even if attacker can watch the message between client and server pass over the network, the attacker still can’t learn the contents of message between client and server.

* 1. *Initiate a Conversation Pretending to be Client*

The project has one-way authentication, so it can’t verify the client.

* 1. *Lie in Wait at Server’s Network Address and Accept a Connection from Client*

The server will use certificate signed by CA to verify itself.

* 1. *Read Server’s Database*

The attacker can get server’s private key to decrypt the master key from client.

* 1. *Man-in-the-Middle Attack*

The attacker can’t get useful information if he doesn’t know server’s private key or master key.

1. **Issues and Solutions**

During the development process, we discover some issues that we did not think they would surface. Below is a list of issues and our according solutions:

* 1. *Spoof FIN Attack and IP Hijacking Attack*

In our original design, there is an issue that anyone who knows the server’s IP address can connect to the server, even though server is currently in a connection with another (legit) client. Among those clients, if any of them quit, the server closes the socket, thus all other clients have to end the connection as well. We fix this issue by introducing a state checking in the server. After the first client successfully connects, the server is now in “busy mode” and will not accept any other requests. Now no one can interfere with the legit connection.