Report

Z5257079- Zhilin Xie

The realtime chat application is a messaging application which is built on both client-server as well as peer to peer architecture design. Both of which runs on top of the TCP/IP protocol. The project is built on python 3.

Application layer protocol

The application layer protocol designed is basically a stipped down version of the HTTP protocol in which only includes the route and the body.

`[200] {}`

`[400] _error_message`

`[login] {'username': _username}`

the above examples are examples are the three main ways in which the data packets are sent. The information inside the square backets are either the route or response code, and the information that follows the square brackets can be data in either a string or a json format.

Architectural design

The Server Utilises a multithreaded design which creates a new thread for each client, whereas the Client utilises a non-blocking I/O design for both its user input, server response as well as P2P connections.

Client Architecture, Incorporates a select statement, which takes in a list of readables, which includes :

- sys.stdin
- clientSocket(connection to server)
- P2PSockets (connections to other clients)

the client utilises a select statement which blocks until either one of the readables return data. i.e. either the user inputs a command to STDIN or the server/other client connections sends it a packet, in which it will then handle the respective input accordingly.

Design

The program utilises both procedural as well as object oriented designs. The three main classes created belong to the server. Clientthread, Timer, User.

Clienthread is a thread that represents and controls all client server interaction for a single client, and is responsible to all server interactions for that client.

Timer class is a separate thread which acts as a daemon countdown timer and is created for each Clienthread, to model timeout for the specific client. The choice behind having a separate thread for it is so that we can call a reset function whenever the client sends a valid command to the server.

The User class models the specific user and user details associated with the user who has logged in to the client and is stored in the global store for all users.

Storage

There are three main global storages, the ALL_USERS storage, the ONLINE_USERS storage as well as the BLOCKED_USERS storage.

ALL_USERS is a list of User objects, which is added to the global storage when a new user is registered

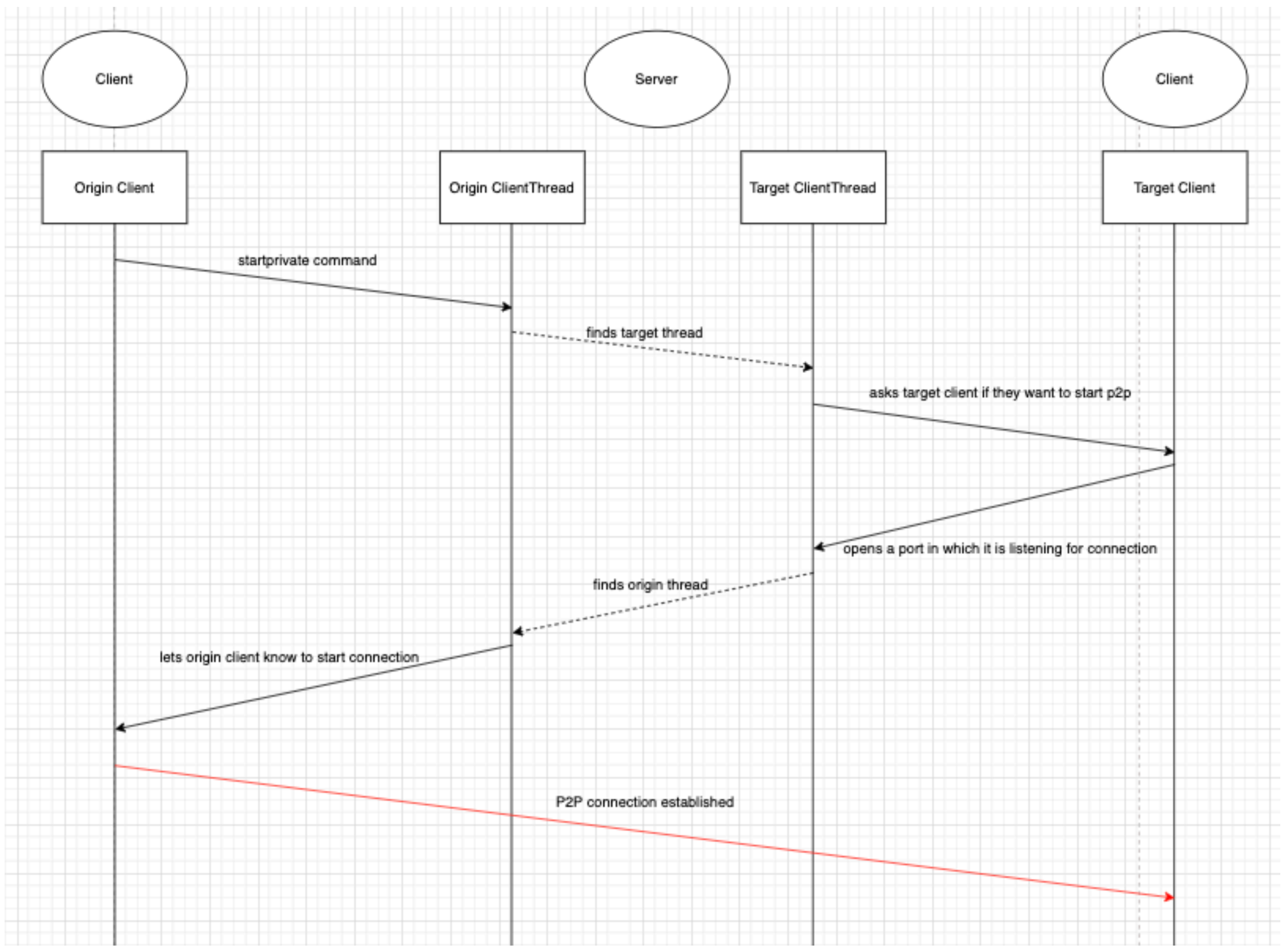
ONLINE_USERS is a list of ClientThread Objects, (NOT USERS), The design choice for this is obvious, we want to directly be able to interact with different Clients via their live ClientThread instances, and we are able to check which user it is because the ClientThread object stores a User object as an attribute.

BLOCKED_USERS is the storage for currently banned users, which the server checks to see if the user is currently banned because they entered too many incorrect passwords.

P2P architecture

For the P2P connection to be able to establish, the origin client sends the target client a request asking to start a private connection.

In this example below the target client accepts the request.


Client
Server
Client
Origin Client
Origin ClientThread
Target ClientThread
Target Client
startprivate command
finds target thread
asks target client if they want to start p2p
opens a port in which it is listening for connection
finds origin thread
lets origin client know to start connection
P2P connection established