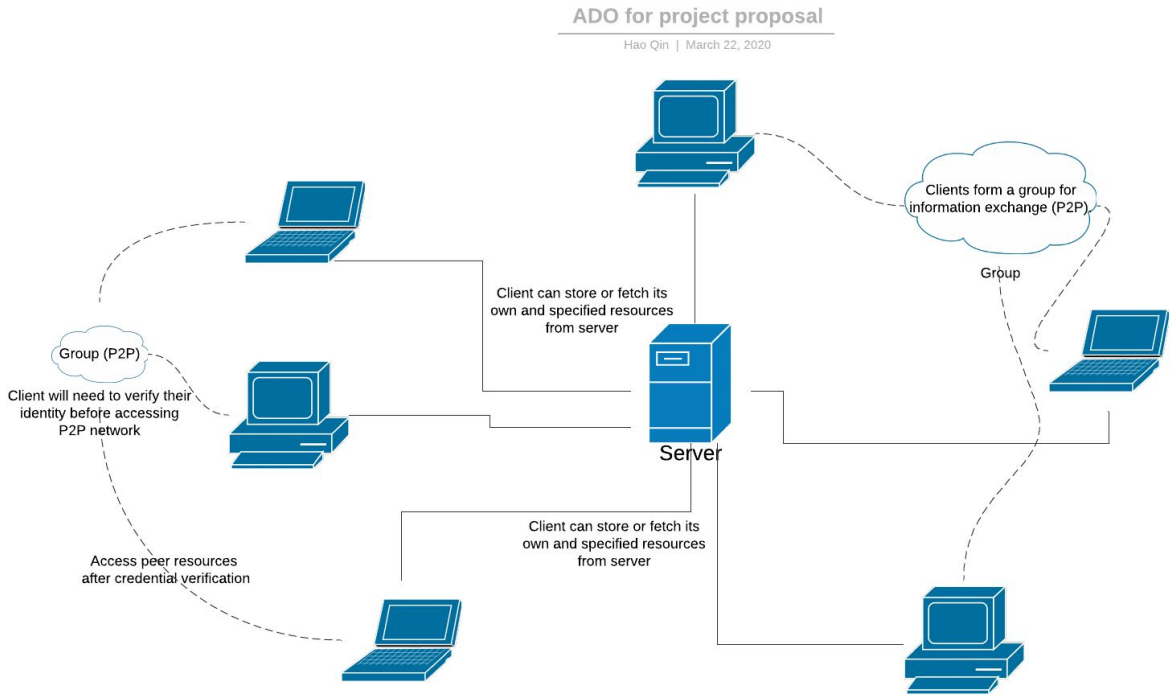
Project Proposal

Team members: Zhenhui Guo, Wanjia Tang, Hao Qin

Summary description

We would like to propose a peer-to-peer version of twitter which has strong guarantee of privacy and anonymity. Unlike the current twitter where clients can post public content to all twitter clients, this peer-to-peer twitter system adopts the notion of groups: one client can create a group and an access key of the group. The access key is shared to other clients, who can join this group through this access key. When a client makes a post to all other clients, it sets the group which has access to this post, and only clients within the right group will be able to access the post. Clients outside of this group will not be able to see the content. If someone is spying on the group, it cannot know the content of the post because the post will be encrypted.

Architecture overview diagram



Design description

The system works as follows:

- We have one or more server to maintain information about client login credentials, and we also have several servers which are databases to store encrypted client posts
- A client should first register itself by providing its clientname and password to the server. The server will store the login credentials in its database and when a client tries to login, it checks if there is a match in its database. If yes, login attempt succeeds, otherwise login attempt fails

- The server marks the client by two states: active and inactive. A client is marked as active by the server once it is logged. A client will be marked as inactive when it logs out. The server will periodically send requests to all clients to test if it is still active. If the client's response does not reach the server within a certain time limit, this client will also be marked as inactive
- Once a client is logged in, it can:
    a. create a group: a client can create a group by providing a name and a password to the group. At the time of creation, the server will record the group name and the current group member(the client who creates the group)
    b. join a group: A client can join a group by first sending the group name to the server. The server will verify if the group exists or not, if yes, it sends the list of active group members to the client and the client sends the group name and password to one of the group members to get verified.
    c. pull group posts: when a client is logged in, or when a client is added to a new group, it will automatically pull all the previous posts within the group. The client will request such information from the server. The server, upon receiving the request, will send encrypted group posts data to the client, and the client will decrypt the data with the public key of the group
    d. send posts within a group: a client can send a post within a specified group. The client must be a current member of the group. The client will first encrypt the data with a private key and send it to the server for backup. Then the client also sends encrypted data to all current active members of the group
    e. opt-out the group: a client can choose to exit a group. Once it does so, the server will delete it from the member list of the group; the other group members will also remove it from the group list; the public key of the group will also be erased from the group, and it will lose access to all the posts of the group
    f. logout of the system. This stops the client from receiving posts from any group. The server will mark the client as deactive

Implementation approach

As discussed above, some important functions of this peer-to-peer twitter system includes:

- Data encryption: we will use a RSA algorithm to generate a public and private key for each group. The client which initializes the group will run the algorithm to generate the public and private key, and all other clients which joined the group, will be informed about the two keys. When sending posts within the group, each client shall encrypt the data with the private key and then send the encrypted data.
- Data communication: The network layer of data transmission will be based on TCP, which is reliable transmission. Client-to-client and client-server communication will be sent in the form of tcp packets. Also, since there's a one to many communication pattern(a client sends posts to all current members of the group), we can adopt the group communication strategy to use multicasting.

We will have the following components in the system:

- Server(s) to hold client login credentials. There should be a central server which maintains a database for client login credentials. When a client registers itself with this twitter system, the central server puts its clientname and password into its database; when a client tries to login, the server checks if there is a record for the clientname and password it provides. Further, to handle scalability and increase fault tolerance, we can extend this to multiple servers which maintain replications of the same database with the two phase commit protocol.
- Servers to store encrypted data of client posts. The communication is peer-to-peer, meaning clients can communicate without the server transmitting the data. However, we do need servers to store the posts a client has posted. When a client is about to post a message, it sends its encrypted data to the server, and the server stores the information. Since the data is encrypted, and the server does not know the public key, it will not know the content of the data, which ensures privacy. This data storage system should also be distributed: the posts within the same group will be stored across a series of servers. There may be two ways of data storage:
    - 1. We can make replications of posts and store replications on different servers, so that each server maintains all the posts within the same group. Since the client needs to pull all the posts within the same group. Using replications will make it easier for this operation: a client will just send a pull request to one server and it gets all the information.
    - 2. We do not make replications, each server just stores a portion of the posts within a group. If we do not use replications, the server will need to query other servers to get all posts. We have not decided which one is a better practice.

Key algorithm involved

- time and clocks: each post will be given a Lamport timestamp(Lamport clock or vector clock). Since in the real twitter system, posts are arranged by time, this is also needed in our peer-to-peer twitter system. The clock on each client will be increased when it creates a new post, or receives a post from other clocks. On each client, the posts will be arranged in time of timestamp
- group communication: since a post is made to all members within the same group, this is apparently a pattern for group communication. We can use multicast for this operation
- distributed mutual exclusion: there's several points where we need distributed mutual exclusion: 1. when multiple clients login at the same time. This is relatively easy to handle because the server only needs to read from a database. A shared lock should be enough. 2. when two clients try to make posts at the same time. We can either use the central server algorithm or the ring-based algorithm to make sure that one client makes the post first, and then the other client makes the post. 3. When a client logs out and another client makes a post. We should first let the client log out, and then the other client logs in
- Managing replicated data: there may be two points where we need replicated data: 1. The central server which holds client login credentials. To handle scalability, we will need multiple servers holding replications of the same database which stores user

login credentials; 2. The server which records client posts. This is still under discussion, and one possible way is that each server maintains the same replications of all the posts within the same group.

- P2P Network: The twitter-like application that we will create shall support p2p connection between nodes which does not include the use of centralized administrative systems. For now, we decide to build structured system data to make P2P more efficient. However, we will consider which method we shall use for decentralization during our implementation, since only during implementation we will know which way is better for our application.
- Consensus: The servers that maintain the same database will be replicated among each other, and all servers will need to keep a consensus when one server is updated. This will be achieved by asynchronously updating each server's data and reaching an agreement about when to commit to a change and when to abort.

Expected results

The system should have the expected behaviour:

- The are two kind of servers in this system: one to keep user login credential and keep active client list, the other to keep client posts
- The system is robust to failure such that any client, if failed, can retain all the posts after it is reconnected. Any server, if failed, can retain the data from other servers after it is reconnected.
- The system is highly peer-to-peer. Most communication happens between clients. Client-server communication is mainly to keep the current system status.
- This system delivers safe and private message exchange among groups with encryption, and it is decentralized in ways such that there is no central server could spy on what communication is going on in the group.