Design Document

**SUMMARY**

This project was a twitter service that uses grpc to connect clients to a server. Clients could: create users, follow, unfollow, list info, and show a timeline. The server was the program that was actually executing all this code, all the client would do is connect to a server and send commands/messages to the server.

**HOW TO RUN:**

Make the project: make

Start the server: ./tsd -h [hostname/ip] -p[port number]

Connect a client: ./tsc -u[username] -h[hostname/ip] -p[port number]

Note about VM: we used virtual box to test our server client connections and we had to do extra for it to work. Not only do we need the NAT NETWORK and different MAC addresses. We also had to add a new network adapter “Bridged Adapter”. Then to find the ip address to use, for each VM, we went into the terminal and ran the command ifconfig. This listed all devices that were using internet, we used enp0s8. The programs will not be able to communicate otherwise, this is also only for windows 10, I have no idea how to do it for macs.

**SERVER**

The server contained eight main sections/key components. The first five are the functions mentioned above in the summary, the other three are a user object that stored information about an user, a is a container for those user objects, and the persistence. Let’s discuss the five functions first.

First, we will discuss the container and user object. The container itself is a map with a username as a key and the value is a user object. The user object contains four things:

* username:
  + this is just the username of the logged-on user
* follower vector:
  + this is a vector of users that are following the user. We chose to implement it this way instead of having a vector of people that you follow because when writing timelines, it would be easier to send messages out to one vector of users rather than iterating through multiple vectors to see if someone is following others.
* timeline vector:
  + contains timeline objects which are the username that posted, the post, and the time of the post
* timeline stream object:
  + this is the grpc channel

Now to discuss the functions:

* create\_user:

This is obviously used to create a new user, this is called when a client first connects to the server, if the username is already in use the server will return that the username is already in use and the client must reconnect with a new username. Then the user is added the container.

* follow:

The follow command does as it sounds like, after logging in as a user that user name can choose to follow others. If the username that is to be followed does not exist or is already being followed by you then errors are returned. If no errors occur then the user gets added the list of followers.

* unfollow:

This is practically the same as follow the only difference is that it first makes sure that the user is following the user they wish to unfollow. Then they are removed from the other user’s followers list.

* list:

This lists all existent users in the map and the followers of the user making the request.

* timeline:

This is the timeline. If a user logs on and enters timeline mode the most recent 20 messages will be displayed, if there are not 20 then the most recent < 20 messages will be displayed. In the function itself there is a w­­­riter stream that listens for inputs from clients. Once a post is received it iterates through the list of that client’s followers and adds the post to their respective timelines.

Persistence:

The way we persist to and from the server is at initial startup and when the server shuts down. The information that is stored is the same that is in the map, but it is represented in a json file called timeline\_database. When starting the if there is information available the server uploads it to the container. Then when a user that is already in the map connects their grpc stream object is added. On shutdown everything is copied over to the database except the stream object.

**CLIENT**

Connection to server:

To connect we use grpc protocols, by creating a channel using a hostname and port and using this channel to create a stub pointer. After creating the stub, the program sends the username provided by the command-line arguments (defaulted to ‘default’) by calling the ‘create\_user’ function on the server to register or login the username.

Processing commands:

This is just simple parsing, if the user wants to: follow, unfollow, list, or show timeline mode, a menu is prompted with those options. From the channel the client then calls the server to execute whichever function it wants, takes the reply object from the server and populates an IReply object so that client.h can handle the reply accordingly.

Processing Timelines:

The client has two threads, a read and a write thread, the reader thread is messages from the server that other users that you are following (or yourself) have posted. It records time, username, and post of the tweet and displays it on the screen. For writing we send a username, post, and time to the server and the server sends the messages out to other clients.