PROJECT DESIGN

**Nathaniel Leake | Alex Kaiser**

CSCE 438, HW3.1

**Overview**

This project utilizes GRPC (Google’s Remote Process Communication) handle to build a miniature client-server social network. The basic functionally includes commands to FOLLOW, UNFOLLOW, and LIST users, and to show timelines. The social network is spread across multiple servers (meaning if any server other than the routing server dies, the network will be okay) and all data is persistent between reboots of the network.

**Detail Execution Instruction on GitHub:**

<https://github.tamu.edu/nateleake/438-HW3>

* **Clientside commands**
* FOLLOW [username]

This will result in all timeline messages posted by the user you followed also appearing on your own timeline

* UNFOLLOW [username]

Effect is opposite of FOLLOW command; posts from the specified user will no longer appear on your timeline

* LIST

The output of this command is a list off all users using the TSN, along with a list of your followers.

* TIMELINE

Enters timeline mode. Once the user has entered this command, the only way to exit is by terminating the program. When entering timeline mode, the user will be shown the 20 most recent posts on his/her timeline. While in timeline mode, a user can write new posts, which in turn will add those same posts to his/her followers’ timelines.

A bonus feature included with out timeline mode is a live-feed of posts from the users one is following.

* **Serverside**

The Available Server is the one that actually processes all these commands. When a command is received from a user, the Available Server validates and processes that command, generates and returns Reply and Status objects. If the Reply is of the type SUCCESS, the Available Server also forwards the command on to all the other (redundant) servers so that they can keep their databases updated as well. If any of the other servers happen to be down, the Available Server will keep a queue of commands that server has missed so that if it comes online again it can be brought up to speed. This queue of messages is NOT transferred to the new Available Server if/when the current Available Server dies, so it is currently possible to lose data on certain servers if you play the adversary correctly.

* **Handling Dying Servers**

When the available server dies, the other servers will send an Election Message to all available master servers in the area. The other servers will respond to the election message with either nothing if they are crashed, or their PID if they are alive. The new coordinator will be elected by choosing the newest server with the highest PID. In the background, the slave server will be hard at work attempting to restart the dead server. Furthermore, we will use boolean arguments to distinguish between whether the server is the router or not.

* **Transfer of Data Between Servers**

In order to maintain persistent data between all the available servers, the active server forwards all data to its fellow master servers after a successful command has been sent to the coordinating server. However, crashed servers do not remain persistent as following the project guidelines.

* **Details & Extra Information**
* The default port for the routing server is 12021
* The default port for the master servers is 12022
* The default port for the slave servers is 12023
* All data on a server is saved on shutdown in server\_data/
* Don’t do LIST when there are a lot of users =P

**Refer to the README for more extensive details on how to run the program**

Basically, to run the program, type make,

* bash start\_normal.sh (to start a master-slave pair)
* bash start\_router.sh (to start a router-slave pair)

**Note:**

To kill a process, run kill PID, where PID is the id of the master server. The slave server will restart the master server in (usually) under a second. Master servers always have a lower PID than their respective slaver servers.