CSE 452/CSE M552
Project 1: Client-Server Twitter

One page design document due: 5pm, April 11, 2013

Assignment Due: 5pm, April 23, 2013

The project is to be done in groups of two to three. First, please form a group, and send mail to the TA's to let them know your group has been formed and who the members are.

The project has three parts. The first part is to build a simple client-server Twitter application. For this assignment, you may (should!) assume that there is only one client and one server. We will relax those assumptions in a later assignment, first to provide transactional semantics to Twitter state updates and then to support highly available service.

1. Message Layer

We will provide you with some basic Java code to implement a reliable, in-order message layer, that can be either linked with a simulator or an emulator to test your code against realistic conditions such as packet loss and node failure. When a node sends a message with this layer, the layer uses timeouts to resend packets until they are acknowledged by the receiver. The receiver then uses a sequence number to hold on to out-of-order packets before delivery. This is somewhat similar to TCP's support for reliability and in-order delivery, except that a theoretically infinite number of packets can be in-flight without acknowledgment.

It is important to note, however, that this layer is not complete. Specifically, it is not correct in the presence of node failures (you should think about why this is true). Since reliable, in-order delivery is useful for this project, before implementing the rest of this assignment, you should correct this problem.

2. Named File Storage

The next step is to use the message layer to implement a way for the client to store and retrieve named data from the server. Since your code will be the only user of these routines, you can (should!) make any simplifying assumptions that you would like, e.g., that there are only a handful of file names, or that files are read and written in their entirety.

For example, you may want to implement the following routines (feel free to change this list as you flesh out part 3):

Create filename: creates a file name on the server

Read filename: reads a file from the server

Append filename: add data to the end of a file

Check filename date: has file been changed since previous version?

Delete filename: deletes an existing file from the server

A requirement is for you to develop a common design pattern that you use across these various routines, e.g., a remote procedure call (RPC) system with some well-defined semantics for failure recovery.

3. Twitter Operations

The Twitter client does a set of twitter like operations. Of course, at this stage, there is only one client, so you can think of it as designed for a somewhat egotistical person, but we'll fix that later.

You should feel free to make the user interface anything you would like (text command line is fine). The operations your system should support are:

Create a user
Login/logout as user
Post to a twitter stream
Add/delete a follower to/from a twitter stream
Read all (unread) posts that a user is following

These operations are semantically ordered – e.g., a post to a stream that occurs before someone is added as a follower is NOT delivered to the follower. While you could implement these operations directly using RPC, we encourage you instead to use your storage system, as that will make later assignments easier to implement. A requirement later on is that all of the application logic will be done on the client, and only changes to state are reflected to the server. The reason for this will become apparent in the subsequent assignments.

4. Turnin

We will use the standard Catalyst turn-in toolset. The assignment has two separate deadlines. The first is a one page design document, due approximately 1.5 weeks before the main deadline; we welcome earlier turn-ins, but if you do that, please let us know when we should look at it.

The design document is graded Satisfactory/Non-Sat. The goal is to help us debug your design, before you spend a long time on it. A quick sketch is enough. Please go into **only** enough detail to allow us to know how you are planning to solve the assignment.

For the regular turn-in, please provide a tar file with code, a design overview, documentation sufficient for us to understand your code, and the result of some test runs that are enough to convince you that your code is working. The design overview is a longer version of the design document turned in earlier – you can think of it as a high-level guide to someone reading your code.

During the week the assignment is due, we will also schedule 15 minute demos with each group. These will take place instead of section during those weeks. Your grade on the assignment will be based on **both** the demo and the turn-in.