Design Exercise 1
Engineering Notebook

Adarsh Hiremath and Andrew Palacci

February 19, 2023

Introduction

In our design exercise, we implemented an end-to-end client-server chat application, first with our own wire protocol and later with gRPC. We built our chat application using Python with a simple tutorial and eventually built functionality on top of our minimal chat room application to fulfill the assignment specification. As currently implemented, our chat application supports the following:

- Account creation, login, deletion, and deactivation.
- Sending messages between accounts, even when some accounts are offline.
- Listing or filtering all users who have created accounts.

The source code for our chat application can be found here. Throughout this notebook, Andrew and I intend to describe various design decisions made throughout the development of this project.

Programming Language

Initially, Andrew and I considered 3 programming languages to implement our client-server chat application in: Python, Java, and C++. While Professor Waldo recommended that we complete the project in Java, Andrew and I decided to use another language since we couldn't find an elegant solution for handling different versions of the Java Virtual Machine (JVM) during peer grading.

Ultimately, Andrew and I decided to implement our design exercise in Python because of the extensive documentation for the socket module. While C++ was the clear choice for low-level optimizations, Andrew and I wanted to pick a language that would be most beneficial for learning. After some exploration on the internet, it became clear that tutorials for socket programming in Python were far more accessible to us than tutorials for socket programming in C++.

The final benefit to using Python instead of C++ was code reproducibility. Most if not all students have Python natively on their machines. Any student can run our program with a simple command allowing them to mirror the dependencies listed in a requirements.txt file, regardless of their machine. Conversely, maintaining code reproducibility across machines for a C++ implementation would require creating precompiled binaries for students, which we found to be far more tedious.

Server vs. Client

Before implementing our client or server, Andrew and I spent some time considering what input processing we wanted to put on the server side versus the client side. As will be explained later in this writeup, our wire protocol handles information differently depending on the operation code provided before a client types in the delimeter. Our options were to either process the operations on the client side, rejecting invalid inputs before even pinging the server, or handling the operations on the server side.

Andrew and I decided to process information sent by the client on the server side for a variety of reasons:

- A client side approach to input validation requires all clients to download updates if any operation specifications change. This results in versioning issues.
- Validation done on the client side results in code redundancy since every client has duplicate code specifying what inputs are allowed.
- Clients having access to the type of operations they can use to ping the server results in security issues.

Wire Protocol

Here, we will specify the functionalities supported by our wire protocol as well as provide further explanation about how our protocol was implemented. Our wire protocol utilizes a delimeter system to distinguish between different requests from the user. Our delimeter is '|', so any requests made to the server must include the appropriate command followed by the delimeter followed by any relevant parameters. For example, sending a message to another user would look something like:

's | <recipientUUID> | <message>.' Below, we've included a complete list of the arguments accepted by our wire protocol.

  - Create an account. Usage: c | <username>
  - Log into an account. Usage: l | <username>
  - Send a message to a user. Usage: s | <recipient$_{u}sername$ > | $<$ *message* $>$ $Listallusers.Usage : u$
- Filter accounts. Usage: f | <filter$_{r}egex$ > $Deleteyouraccount.Usage : d| < confirm_{u}sername >$