Protocol for Multi-chat Application

Version 3.0

November 2019

Prepared for:

COIS 4310 - Computer Networking

Trent University

By:

Taylor McLean

Table of Contents:

1. Application Summary and Purpose

This application will run through a terminal command line interface to allow for up to 5 users to connect and send messages, either to other individual users or to all users. Both the client and the server can be run on any machine that has Python 2.7 installed, and users can connect from any machine provided they connect to the correct IP address and port that the server is running on. This application was designed for educational purposes, as a “proof of concept”, and to practice computer networking from both a programmatic and protocol perspective.

2. Application Architecture

The application has both a server and client component. The server component will act to process the messages sent by the users, provide services based on the verbs given, and route messages to their corresponding destinations. The client component will provide a command line interface for users to connect to a chat, request services from the server, and send messages to other users in the chat.

A. Server

The server is built on the Python 2.7 interpreter, which implements many of the low-level system calls with respect to network sockets and their communication. IPv4 is used as the internet protocol, while TCP is used as the protocol for data exchange. The server listens on port 57374.

The server will listen for up to 5 connections, and will generate a new thread for each connection. This ensures that the server can listen to each connection simultaneously.

B. Client

The client should also use TCP and port 57374 to communicate with the server. It can use a command-line interface to send messages and requests to the server. The protocol for formatting these messages will be outlined in the following sections of the document.

3. Header Format

All messages sent to the server should contain a header with the following sections, in respective order (note that data will be decoded using ASCII):

Code version number - 3 bytes: This will specify the application version in the format x.x where x is a single digit value. This should not be left blank.

Source client name - 30 bytes: This will specify the client alias that is sending the message. The alias should be left-aligned in the 30 bytes, and any remaining bytes not used by the alias should be filled with whitespace. This should not be left blank. If the user is registering their alias, the source client name should be given as “temp”. Temp cannot be used to do anything but initially register an alias.

Destination client name - 30 bytes: This will specify the client alias to which the message is being sent. Similarly to the source client name, the alias should be left-aligned, with remaining bytes as whitespace. If the message is intended to be sent to all users, this field can be left blank.

Request verb - 3 bytes: This will specify the action that the user is requesting from the server. If left blank, the server will not perform any action.

Encryption technique - 32 bytes: This will specify the technique of encryption used by the sending client to encrypt the sent message. If the message is not encrypted, this field should have the value “cleartext”.

An additional 158 bytes is appended to the end of the header, to be reserved for future modifications to the header format. This makes the header an even 256 bytes, and allows the socket buffer size to be a multiple of two.

Data/message - 256 bytes: This section will contain the message to be sent by the user. Messages on this application can have a maximum size of 256 ASCII characters. The message should be left-aligned and filled with whitespace.

4. Data Format

The data section should be ASCII encoded, and contain a maximum of 256 characters. It is recommended that the client outline the possible verbs that the server will respond to, and allow the user to input their desired verb along with the message they intend to send.

A suggested format is the following:
The user prepends “*verb*:” to their message input; verb being either the alias of the user they want to send the message to, or another verb that the server will respond to (i.e. “reg:”, “all:”, “one:”, “who:”, “enc:”, “bye:”). The client would then parse the verb from the user’s input into the command line interface, and add the appropriate verb and/or destination client to the header before sending the message off.

5. Verbs and Server Functionality

The following verbs can be used in the “request verb” header section to request an action from the server:

reg - register an alias with the server. This will pair the client’s connection with the alias provided. This will need to be done with the first message sent to the server after initially connecting, otherwise the server will not recognize requests from the client. If this verb is provided after the initial registration, it will change the alias paired with the client's connection, and future messages should be sent under this alias in order for requests to be recognized. The alias must be a maximum of 30 ASCII characters. If more characters are provided, the alias will be cut off at 30 characters.

one - this verb is to be used when the message is intended for one individual user. The server will send the message to the user alias specified in the "destination client name" header section.

all - to be used when the message is intended to be sent to all users. The server will send the message to all users that have an active connection in the chat.

who - to be used when the user wants to be sent a list of all active users from the server. The server will send the user a list of all other users with an active connection.

enc - to be used when the user wants to toggle between sending encrypted/unencrypted messages. The client will notify which setting the encryption toggle is on.

bye - to be used when the user wants to disconnect from the chat. The server will close the socket connection.