Assignment 5 Part 2 Memo

Partners: Mark Thekkethala & Kate Rupar

Introduction

Assignment 5 lays out a blueprint for a conceived network layer. This includes servers and clients who are able to directly communicate server to client or client to client. Clients can send data to addresses which they know, and clients all know a master address (server) to which they register. From here the master address, will update all the other clients, so that each client is able to communicate with another client.

Design

The code base is broken into an API for the client and an API for the server. The client's functionality is broken down into dealing with two parts: addressing and data. The client API has methods for sending the update that a new client has been created to the master address or server so that all the other clients know the new client's address for their direct communication functionality. The client is also able to receive updates from a server which is how the new clients will receive the list of client addresses and the current client will receive the newly added client address. The second part of a client is the ability to send and receive data. The client will know the list of destinations that it can send data to, and the method send_data() takes in one of these addresses and the packet of data to send. The recv_data() is the functionality for a client to receive and handle data. The client has a dump_table() method which can be used to return an Object** of all the addresses that this client has saved.

The server API follows a similar set of method naming conventions; however, the functionality differs slightly because a server will serve as the master address. This master address is known upon construction of all clients, so that they can be added to the network. This master address is set in upon construction of the server, and can be retrieved using the get_master_addr() which returns the address as a String*. The server also has a similar method to client dump_table(), but this table should be the most inclusive table, as the server has all the addresses of all the clients in the network. Server's send_update() functions differently than client's send_update() because the client's simply registers itself with the server, while the server's send update sends the table of all registered clients to all the clients in the network. Server's recv_update() interacts with a client's send_update(), by simply adding the network address of the client to the table. The send/recv_data() functions for the server are the same as the client's functions because data needs to be transferred in the same way.

Challenges & Open Issues

Some of the challenges that we still face include the direct communication of two clients and the sending/receiving data in a common type (currently Object*). The first problem of direct communication comes from our understanding of the specification. A client is able to communicate with a server. Our initial thought was to have a client put an destination address on the data as a header and use the server to act as an intermediary to forward data to the correct clients. In this implementation, the server would also act as a router. However, with direct communication between client-client, we need to have one client and the second client act as a server. This functionality has not prototyped out.

The second issue is the data we are sending. Currently, we are deciding how to package the data together where all clients and servers are able to both send and receive/handle the data that comes in. Our current setup needs to be rethought because we had planned on sending data with a header of destination address and data type which would be used to cast the Object type data back to the true data form.