Team Name Name Name

Ryan B., Salvatore B., Chris H., Andrew S.

CS 465

Project 1: p2pchat

Delivered 18th September 2014

p2pchat is a peer-to-peer chat program written in Python 3.2. In theory it can support any number of users, but to preserve performance we have set an arbitrary limit at 15.

Design

Assumptions

Some assumptions are made about the operation of the application. The two largest ones are:

1. Participants in the network will always know (before attempting to connect) the address of at least one peer already in the network, and
2. Participants in the network will never leave without taking care of the proper procedures for doing so, i.e.: any disconnection will be a graceful one.

Initial State

In the initial state, only one peer exists on the network, and that peer maintains a dictionary of all active peers, populated with exactly one item: its own IP address and screen name*. This peer listens on one port for an incoming connection request.

* Screen names are provided by all peers: the program prompts users for a name on startup.

State Changes

- New Connection -- At some point, another peer will make a request to join the network. Again, the assumption is made that the peer wishing to connect already has the IP of the active peer on the network. When this happens, the peer already within the network adds the IP address and screen name of the new peer to its dictionary of peers, and then sends the updated list to every other peer on the network. In the state with only one active and one joining peer, this means the list is now two items long and the joining peer now has a list with its own IP/SN and that of the other peer.
- Message Send -- After making the initial network connection, a peer is dropped into a terminal-based chatroom interface which displays information from new messages as they're received. To send a message, the user types out a message at this prompt and presses Enter. That peer then packages the new input as JSON, forms a Message object with it, and sends the Message over the network to each peer in its dictionary of peers, in turn.
- List Update -- When a user connects or disconnects, it becomes necessary for peers to share information about the state of the dictionary they're holding network state in. They do this by sending messages which contain updated versions of their own dictionaries (again, encoded as JSON). When new lists come down the wire, they're accepted as canonical, and the receiving peer replaces its own list with the new one.
- Username Update -- Users may change their username at any time. When they choose to do so, the peer forms a Message object flagged as a name change, which contains the new name in its body, and

sends it out to each peer in its list. When a name-change request is received, the receiving peer updates the relevant entry in its dictionary of peers.
- Disconnect -- When a peer disconnects from the network, they send a message flagged as a disconnect request to each peer in turn. Upon receipt of a disconnect request, the receiving peer fetches the IP of the sender from the Message it received, finds the entry corresponding to that IP in its peer dictionary, and deletes that entry.

Implementation Details

- Everything is Messages -- All communication about requests and updates between peers, as well as all information about network state, is communicated as Message objects. The Peer.Listen_Helper() method takes in these messages and alters the state of the Peer depending on their content.
- Connection-Oriented, Using TCP -- Communication within the network is done via TCP. When a peer wishes to send a message, it establishes a connection with each peer in its dictionary, sends the message, and then closes the connection. This allows us to take advantage of TCP's reliability, which is critical given the above point (Evertyhing is Messages).