Protocol Specifications
For Project 3.1 COP 5615

By: Stephen Williams
& Chris Pearson

Table of Contents

I. General Message Structure
II. Chat Server Multicast Commands
III. Chat Server Unicast Commands
IV. User Agent to Chat Server Commands

General Message Structure

Each message will be broken into three space separated Base64Encoded Chunks that contain:

<Who you are> <What you want> <The sender’s data>

When decoded, the first chunk will contain all of the necessary information about the sender. The format for that information:

<Alias/Server name> <IP Address> <Port>

The second chunk will contain the command that the sender is either trying to make or if its User Agent to Chat Server, replying from. It will also contain additional information as to the success or failure of the command. The format for that will be:

<command> <success/failure>

The third and final chunk will be the data the sender is trying to pass. Since this is command specific, the structure of each will differ based on the command being implemented.

Chat Server Multicast Commands

Some commands will be sent over a predetermined multicast set up for Chat Server discovery.
**newChat:** All new chats send this command to announce their presence to the other chat server. This command contains no other information than the command and who sent it.

**allRollcall:** A heart beat to the system this command can be sent at any time to get responses from all of the chat servers in the system. This can be used to discover new chat servers when a neighbor goes down, or to establish the life of a non-neighbor. This command contains no other information than the command and who sent it.

---

**Chat Server Unicast Commands**

**welcome:** Sent to new servers upon receiving a newChat command this command begins the “Lets be Neighbors” dialog between two servers. This command is not sent if the Chat server is unable to take on anymore neighbors. This command can also be used after an allRollcall command if the responding server says it has room for new neighbors. This command contains no other information than the command and who sent it.

**welcomeAck:** Sent from a new server to an older server who has requested a neighbor relationship. This command contains a success/failure value. If the value is ‘failure,’ then the data packet following is a string message containing the error code. If the value is ‘success,’ then no other information is sent with the command.

**allRollcallAck:** Sent after receiving an AllRollCall command this message contains in its success/failure value the ability to take on new neighbors. If the value is ‘full’ then the sending server can not take on an additional neighbor. If the value is ‘empty’ then a welcome message may be sent. If the value is ‘isNeighbor,’ then the two Chat Servers are already neighbors. No additional data is sent with this command.

**neighborRollcall:** this command simply requests a response from each of its neighbors. This command contains no other information than the command and who sent it.

**neighborRollcallAck:** this command simply responds to the request from a neighbor for a heartbeat. This command contains no other information than the command and who sent it.

**update:** This command requests an update of users and known servers from its neighbors. This command contains no other information than the command and who sent it.

**updateAck:** This command is in response to an update command. This command contains the success/failure value. If the value equals ‘failure’,
then the resulting data is a string containing the error code and message. If the value equals 'success' the resulting data is multiple chunks of Base64 Encoded Users and Server information. After its decoded the data exists in two chunks:

<Neighbors> <User List>

These chunks can be decoded into small Base64 encoded Aliases

<Server A> <Server B> <Server C> ... <Server N>  OR
<User A> <User B> <User C> ... <User N>

Each Alias when decoded looks like the first part of the incoming message

<Alias> <IP Address> <Port>

serverShutdown: This command is sent during a graceful shut down of a chat server. It alerts the chat server’s neighbors that is no longer going to be on, and that they should discontinue sending messages to it. The data for this message contains the final list of neighbors and users. It follows the same format as the updateAck command.

search: This command is sent when a chat server is looking for a particular user. The data for this command contains a single string that is the alias of the user the server is looking for, along with a base64Encoded stack of the servers the command has already visited. It should look something like:

<Alias> <Server List>

The Server List contains the names of the servers the message has come from:

<Server A> <Server B> ... <Server D>

If the receiving server knows the user, then it removes the first user in the list, and sends to that server the searchAck command.

searchAck: This command is sent in response to a search command, once the alias has been found. The data for this command contains a Base64Encoded message containing the information on the user, and a Base64Encoded list of the return path for the message. If no more servers are contained in the return path than the server can pass the message on the requesting user. Else it pulls the next name from the server list and send the searchAck to that server. That data for this command looks like:

<Alias> <Server List>

Where the Alias is decoded into

<Alias Name> <IP Address> <Port>

nameCollision: This command occurs when collision between either Server Names or User Names occurs. The success/failure variable is used. If its value is ‘Server’, then the responding server already knows a server under that name, and the receiving server attempts to change its name. If the value is ‘User’, then the data section contains the name of the user that caused the collision.

Chat Server Unicast Commands
**Search:** This command is sent when looking for a certain user. The data for this command coming from the requesting user contains only the name of the user they are searching for. The return command from the Chat Server contains the success/failure variable. If the value is ‘failure’ the data section contains the error message. If the value is ‘success’ the resulting data is a Base64Encoded list of the users attributes in the format:

<Alias Name> <IP Address> <Port>

**nameCollision:** This command comes from either a collision in the chat server’s own list of users, or from an Chat Server – Chat Server Command. This command contains no other information than the command and who sent it.