Himanshu Gupta
May 7, 2021
CS544-001

Chat Protocol (CP)

1. Service Description

The popularity of online communication tools such as Slack, Discord, etc has been increasing day by day. Our lifestyle has changed to prefer text over other means of communication. With this development, a chat protocol is definitely a good choice to understand and improve different nuisances of modern communication. This paper describes a chat protocol to be used by multiple clients (users) allowing them to join channels, broadcast the message to everyone in the channel, or privately message each other.

This protocol is an application-layer protocol facilitating text communication between the clients. This chat protocol consists of a server and multiple clients. The server is the one making sure received messages are sent to their correct destinations. The server consists of many channels, a channel is just a group of users. Using channels, a user can broadcast a message to everyone in the channel. A user has to first join the channel by authenticating if the server requires a password for that particular user. Once joined, users can retrieve the list of other users in the channel, can message them privately. To make the communication more real-world, users can choose a nickname when connecting to the server. Duplicate nicknames are not allowed. Using the nicknames, users can simply use the nicknames of each other to communicate among themselves. Also, once the user exits the communication, the previously used nickname could be taken by another user.

Furthermore, Once a client's application starts, it sends a request to connect to the server using server hostname and port number (by default port number is 8081). This connection step is the initiation step of the TCP connection between the server and the client. After that, the server checks if that particular user requires a password or not, if not, the server responds with a successfully connected response and a TCP connection established. If a password is required, the server responds with a password-required response, and the user has to provide a password to completely establish the TCP connection. This is also described in Figure 1. Once the connection is established, the user can join the channel by first getting the list of channels and start communicating with others. As new clients connect, this list would get updated alongside new channels.

The format of all the commands and messages is described in section 2. Once client-server connected, messages are exchanged freely, with messages (not commands) not containing any client information explicitly, are interpreted based on the client address sent across the TCP connection by default, attached by the Operating System.

Once the user quits or connection lost due to timeout or due to any hardware failures, the server does not preserve user details such as a nickname. To create, edit permissions of channels, only users with admin rights can do these operations. Users without admin rights can only join channels to which they have permission. Only admin can assign users based on their IP address can assign them to channels, however, admin can join whatever channel he/she likes.

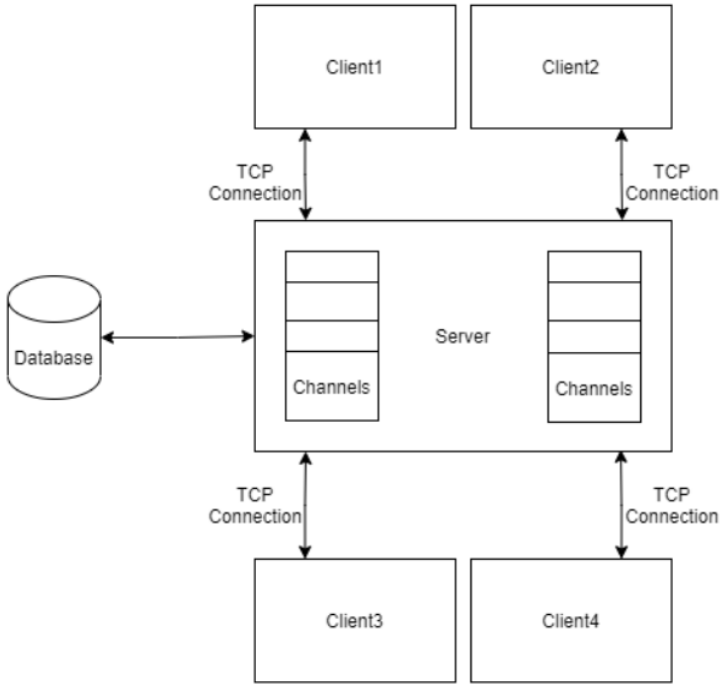

Figure 1: The Chat Protocol Abstract View

With the use of TCP(Transmission Control Protocol) as a transport layer protocol, the chat protocol has all the benefits of TCP. These benefits include reliable end-to-end data delivery with order maintained in which messages are sent/received. It is very important to order messages sent/received as it essentially dictates the QoS(Quality of Services). Also, given the modern demands of the Internet, chat protocol uses IPv6 as its networking protocol. It provides simpler and more efficient routing, better multicast routing, and auto-configuration.

2. Message Definition (PDUs)

The PDU consists of two types, one is the command type and the second is the message type. All the commands type starts with the "\" followed by the command and its params delimited by a space. These params can be optional or required. The commands are not upper-case sensitive. All the PDU ends with the <CRLF>, carriage return, and line feed. Using this <CRLF> criteria, messages are extracted out from the contiguous stream of the bytes,

["\"command]<space>[params]<CRLF>

These “\”, command, space, params, CRLF are the string abstraction in any programming implementation as ultimately would be converted to the ASCII representation.

All the data first has to be transformed from the internal representation to standard 8-bit NVT-ASCII format on the client-side (sending user). Likewise, all the data received by a user (receiver) first have to be transformed from the standard 8-bit NVT ASCII format to the internal representation of the user receiving the message. This encoding/decoding is done as part of the protocol implemented on the respective application side. After the transformation, bytes are attached to the TCP connection for transmission. The server just sends/receives the data in the 8-bit NVT ASCII format by default so no need for transformation. This transformation of data makes the chat protocol interoperable across different platforms. In terms of format, space is represented by “%x20” and CRLF is represented by “%x0D%x0A”.

In terms of response, the server responds to various commands and messages received by the clients. The response consists of a three-digit number, the appropriate message separated by the space, and ended by the <CRLF>. These responses were again sent in the standard ASCII format and transformed into the internal representation on the client-side to represent responses consistently.

<xxx><space><message><CRLF>

These “<xxx>” response number, space, message, CRLF are the string abstraction in any programming implementation as ultimately would be converted to the ASCII representation.

To make the responses consistent, all the 2xx range responses represent successful responses, for example, 200 message received 210 message delivered to <user1>, etc. All the 3xx range responses represent user info required, for example, 301 passwords required, 310 invalid password, etc. All the 4xx range responses represent quit connection responses, for example, 400 quit requests received, etc. All the 5xx range responses represent error responses, for example, 511 invalid command, etc.

A typical client-server flow is represented below.
The client starts with sending connect command to initiate the connection:
\CONNECT nick@hostname:6667 V1.0 -u
> 205 Version Number Accepted
> 301 password required

These “\CONNECT”, nick, hostname, 6667, V1.0, -u, CRLF are the string abstraction in any programming implementation as ultimately would be converted to the ASCII representation.

In this command, nick represents the nickname of the user requesting the connection, the hostname is the hostname of the server, and the port is the port number on which the server is running (by default is

180). Nicknames have to less than 10 characters. The nickname and the hostname are separated by “@”, and the hostname and port number are separated by “:”. If required, the hostname can be converted using the DNS servers to identify the server’s IP. The port number is optional. Also, the “-u/-a” flags are used to find if a user is an admin(-a used for admin). This flag is optional and by default “-u”. “V1.0” represents the version number of the protocol, it is used by the server to handle commands according to the version number. The inclusion of version number also guides the communication configuration after the connection is established between the server and the client. The response is received with 301 password required if password required. If no password received, the response would be 220 connection established.

If a password required, The client would be sending the password using the command PASS. If the password is valid, then 220 response would be sent, otherwise 310 response would be send representing an incorrect password, in this case, the user can re-send the password using the PASS command.

\PASS <password>
> 220 connection established

These “\PASS”, password are the string abstraction in any programming implementation as ultimately would be converted to the ASCII representation.

Once the connection is established, the user can retrieve the list of users or channels using the command LIST with the appropriate flags. Flag “-u” would get the list of active users, whereas “-c” would get the list of channels.

\LIST -u
>abc1
  Bvn23
  Pwg98

These “\LIST”,-u are the string abstraction in any programming implementation as ultimately would be converted to the ASCII representation.

Now, the user has the option to join a channel or message some other user privately. The join of the channel is done using the join command.. If entered channel name not exists, the 551 response would be sent.

\JOIN <channel_name>
> 221 <channel_name> joined

These “\JOIN”, <channel_name> are the string abstraction in any programming implementation as ultimately would be converted to the ASCII representation.

The private message is sent using the msg command. The user name could be another user’s nickname or the IP address of the other user. The entered user_name does not exist, the 550 response of user_name not existing would be sent.

\MSG <user_name> <message>
>210 message delivered to <user_name>

These “\MSG”, <username>, <message> are the string abstraction in any programming implementation as ultimately would be converted to the ASCII representation.

Once the user has joined a channel, the user can broadcast the message to all other users in the channel. Users can simply send a message by just typing the message. The established connection would automatically send it through. If successfully sent, the 200 responses would be sent. Otherwise, 510 responses for the message not sent due to connection error, etc.

\MSG <message>
>200 messages sent.

These “\MSG”, <message> are the string abstraction in any programming implementation as ultimately would be converted to the ASCII representation.

The user can disconnect from the channel using the part command. In this command, channel_name is optional. The response would be sent with the 230 response. Otherwise, if there is some fault, then a 5xx response would be sent.

\PART <channel_name>
>230 <channel_name> exited

These “\PART”, <channel_name> are the string abstraction in any programming implementation as ultimately would be converted to the ASCII representation.

After communication, a user can exit the connection using the exit command. The response would be sent with 250 response.

\EXIT
> 250 connection exited.

There are commands that are only valid in admin mode, some of them are given below. Only admins can create new channels, add permissions for users for a channel. Add a user can be done using add command. Remove a user can be done using the remove command.

\ADD <user> <channel_name>
\REMOVE <user> <channel_name>
These "\ADD", "\REMOVE", <user>, <channel_name> are the string abstraction in any programming implementation as ultimately would be converted to the ASCII representation.
Some other important commands are:

\VERSION -> gives the version number of the protocol

3. Deterministic Finite Automata (DFA)

The Deterministic Finite Automate (DFA) for the chat protocol is described in Figure 2. Each box represents the state of the protocol and the directed arrow represents the state transition. Text over the arrow represents the messages send/receive. Response 2xx or 3xx represents the appropriate message order. Application close or Time out would close the chat connection from certain states.

IDLE: This is a fictional state representing no connection between the server and the client.

REQ_SENT: This state represents the state of the system where the client sent the connection request and waiting for the response, and the server is waiting for the request to come in.

REQ_RECEIVED: This state represents the state of the system where the server responded to the connection request, and the client waiting for the response.

CONNECTED: This state represents the state of the system where the client received the connection established response, both server and client moves to the connected state.

CONNECTED_ADMIN: This state represents the state of the system where the client received the connection established response, both server and client moves to the connected state in the admin mode, the user has to have admin rights.

CLOSE_CONN_REQ_SENT: This state represents the state of the system where the client sends the quit response and waiting for the response. The server still has to process the quit response.

ON_CLOSE_RESP: This state represents the state of the system where the server process the client's quit response and sends the appropriate response back. The client is still waiting for a response back.

CLOSING: This state represents the state of the system where both the server and client have their quit response and request respectively, then close the connection after a time-out.

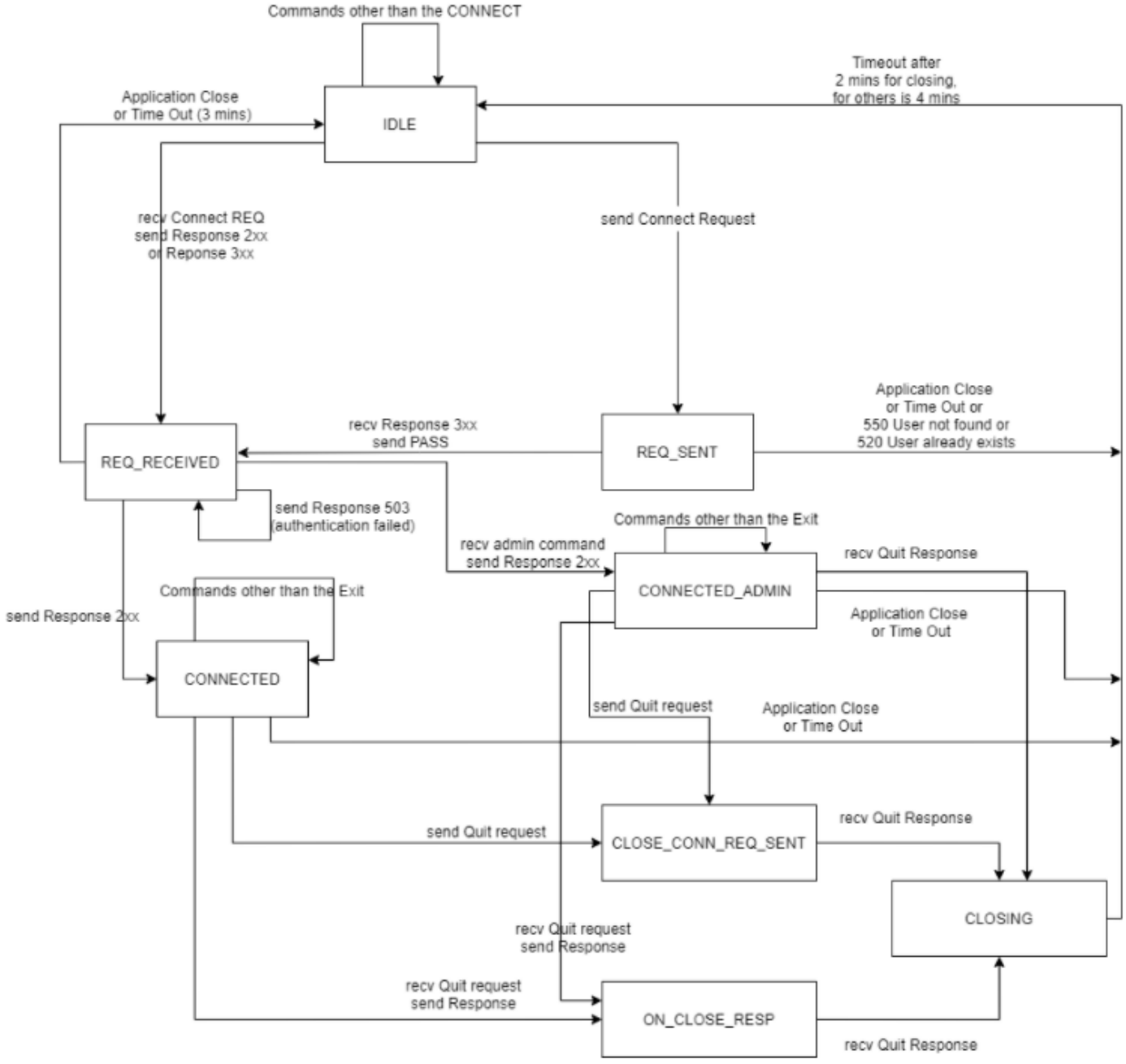

Figure 2: Deterministic Finite Automata For Chat Protocol

4. Extensibility

To cope with future demands, the chat protocol is designed in a way to adds functionalities without making significant changes in the interaction methods. The chat protocol includes a version number so that client and server can use the chat protocol according to the version number, for example, connect command takes a version number to identify the version number. A client can also use version commands to check the version number and ensure the protocol methods. The version number is checked on the server-side, if one of the accepted versions, then returned 205 responses, otherwise invalid version number.

There are several optional fields used in the commands which are chosen optional for further extension in the future. For example, in admin mode, the user can add other users using add <user> command, in the future, add command could accommodate multiple users in one single command.

The use of standard format encoding (ASCII) ensures better encoding, normalization, and comparison in any extensions of platforms. With one standardized format, it is also easier for us to build standardization libraries that support multiple platforms.

With no limitations on command length, extensions can include several commands of different lengths in the future. This ensures the command field could never get exhausted. Also, by using the <space> as a standard delimiter, it is easier to include more parameters and supports more options in the future. These parameters could also include vendor-specific parameters.

With the chat protocol, the command structure is not limited by the bits except the command has to end with the <CRLF> and starts with the "\". For example, in the future, file transmission can be added using commands like \send <user> <filename>.

With that chat protocol to be application-layer protocol, protocols could be added below the chat protocol to support cryptographic agility. The connect command uses the various flags to select the functionality, similarly, a flag could be added to choose the cryptographic algorithm for that connection.

Moreover, with the admin mode functionality, we are making the admin a third-party trust manager. This admin can add/remove users from the channel in the current implementation. Likewise, in the similar fashion of ADD or REMOVE commands, passwords and other information could also be handled by the admin.

5. Security Implications

The chat protocol uses TCP in the transport-layer protocol, this ensures a three-way handshake to ensure the correct user and server connection. Alongside the TCP, authentication also includes a password if required by the server, this password requirement makes sure that no other user can log in by spoofing

the IP address. The user is only allowed to re-try for the invalid passwords, then the admin user has to re-add that user in case of IP spoofing. This makes users more secure about logging.

With the individual TCP connections, messages are ensured to be sent and received by the correct user in the correct order. Even in the case of broadcasting, messages are only broadcasted in the channel specified rather than to all the users.

Only Admins set the permissions of a channel for other users. This makes sure that users not having permission could not enter the unauthorized channel. Channels act as boundaries between the groups of users which are ensured by the chat protocol. Likewise, admins intervene in certain situations where security risk is violated such as exceeded password retrying limit, etc.

To save the data transmission from the packets sniffing, TLS (Transport layer security) is added on top of the TCP layer to protect user info such as passwords, messages, etc. TLS would encapsulate the payloads using encryption algorithms. It is important to maintain the Quality of Services by providing encryption for sending/receiving messages, especially for private messages.

Moreover, the DFA states are rigorously checked and handled errors accordingly such as in the case of admin mode, the admin can not add/remove the user from the channel if the user is connected or already joined the channel, the user can not join other channels without permission.

6. Changes:

1) The Port number has been changed to 8081.
2) Mentioned the underlying transport layer that is TCP.
3) Added more description about the PDU subtypes as discussed in our meeting.
4) Especially added what subtypes should be in any implementation.
5) Implemented the fixes according to the changes suggested in the DFA (mainly the change in response types and commands validity from each state).
6) Updated DFA states definition and diagram.
7) Added more information in the extensibility and the security section.