CS 594 Thomas Salata
Internet-Draft Portland State University
Intended Status: IRC Class Project Specification November 29, 2018
Expires: May 29, 2019

CS 494/594 IRC Project
draft-irc

Copyright Notice

Status of this Memo

Abstract

This memo outlines the protocols used for an internet relay chat application. This application’s development was completed

per requirements listed from Portland State University’s Internetworking Protocols course (CS 494/594) in Fall term of 2018.

1. Introduction

This memo proposes a number of protocols and messaging format for an IRC application that utilizes a client-server model, where clients connect to a single server instance. Clients are then given the option to create and join chat rooms, allowing them to then send messages to other clients in those chat rooms in addition to other commands for requesting information from the server.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119].

3. Message composition

3.1 General message components

```
data {
     “status”: message_status,
     “username”: username,
     “message”: message
}
```

3.1.1 Message type

The message data packets will be passed as JSON objects with UTF-8 character encoding.

3.2 Message fields

- message_status - 4 byte bit-field describing the type of message being sent. Most significant bit MUST be set if the message is from the server and MUST NOT be set if from a client. If the message_status data type is larger than 32 bits, then the most significant bit MUST NOT be set and instead the

32nd bit from the right MUST be set if it is from the server. The message_status field MUST be at least 32 bits in size.

- username - Username specified by client. Messages from server SHALL preserve the username when forwarding from client to client. Messages originating from server SHALL have a username of “SERVER” only. This field MUST always be null terminated, allowing client usernames to have a maximum of 31 characters.

- message - Contents MAY be of length zero depending on message type.

- Total size of the data packet containing the message contents SHALL NOT exceed 2048 bytes.

3.2.1 Message header field values

Note that the bit values are denoted using the standard bit-field operator, “>>,” which denotes bit-shifting the value on the left by the value on the right. “|” denotes an “or” operation.

Message type	Bit value
New connection	1 << 0
List all chat rooms	1 << 1
Create a chat room	1 << 2
Join a chat room	1 << 3
Leave chat room	1 << 4
List chat room members	1 << 5
Send message to clients in chat room(s)	1 << 6
Send private message to a client	1 << 7
Disconnect from server	1 << 8
Error - unable to create chat room	1 << 9
Error - unable to join chat room	1 << 10
Error - unable to leave chat room(s)	1 << 11

Error - unable to list chat room members	1 << 12
Error - failed to forward message to clients in a chat room	1 << 13
Error - failed to forward private message to a client	1 << 14

4. Client Originated Messages

4.1 Initial connection to server

The message_status value MUST be set to 1 and the username field MUST be set to reflect the client’s chosen username. The message field SHOULD be left empty. There is no expected response from the server.

4.2 List chat rooms

The header message_status value is set to 1 << 1 and the username field MUST be set to reflect the client’s chosen username. The message field SHOULD be left empty.

4.2.1 Server response

Upon receipt of the message, the server MUST compose a new message with the message_status value set to 1 << 1 | 1 << 31 and the username set to “SERVER.” The message field MUST be a space dilineated list of room names.

4.2.2 Client parsing of server response

The client SHALL print the name of each room, expecting that the room names are separated by a space in the message field.

4.3 Create a chat room

The header message_status value is set to 1 << 2 and the username field MUST be set to reflect the client’s chosen username. The message field SHALL contain the name of the room. The room name MUST NOT contain spaces or ‘#’ characters.

4.3.1 Server response

The server MUST first check if the client supplied room name already exists in the list of current chat rooms. If it does not exist already, the server MUST add the room name to the list of rooms names and it MUST also add the client to that chat room. The server creates a new message where the message_status value MUST be 1 << 2 | 1 << 31 and the username MUST be “SERVER.” The message field MUST include a string that indicates the room was created and that the user has successfully joined it.

If the room name already exists, the message field MUST be set to 1 << 9 | 1 << 31 and the username MUST be “SERVER.” The message field MUST include a string that indicates the room already exists.

4.4 Join a chat room

The header message_status value is set to 1 << 3 and the username field MUST be set to reflect the client’s chosen username. The message field SHALL contain the name of the room. The room name MUST NOT contain spaces or ‘#’ characters.

4.4.1 Server response

The server MUST first check if the client supplied room name exists in the list of current chat rooms. If it does, then the server creates a new message where the message_status value MUST be 1 << 3 | 1 << 31 and the username MUST be “SERVER.” The message field SHOULD be a string indicating that the user has joined the room.

If the room doesn’t exist, then the server creates a new message where the message_status value MUST be 1 << 10 | 1 << 31 and the username MUST be “SERVER.” The message field SHOULD be a string indicating that the room doesn’t exist.

4.5 Leave a chat room

The header message_status MUST be set to 1 << 4 and the username field MUST be set to reflect the client’s chosen username. The message field SHALL contain the name of the room. The room name MUST NOT contain spaces or ‘#’ characters.

4.5.1 Server response

The server MUST first check if the client supplied room name exists in the list of current chat rooms. If it does, then the server creates a new message where the message_status value MUST be 1 << 4 | 1 << 31 and the username MUST be “SERVER.” The message field SHOULD be a string indicating that the user has left the room.

If the room doesn’t exist, then the server creates a new message where the message_status value MUST be 1 << 11 | 1 << 31 and the username MUST be “SERVER.” The message field SHOULD be a string indicating that the room doesn’t exist.

4.6 List chat room members

The header message_status value is set to 1 << 5 and the username field MUST be set to reflect the client’s chosen username.

4.6.1 Server response

The server MUST first check if the client supplied room name exists in the list of current chat rooms. If it does, then the server creates a new message where the message_status value MUST be 1 << 5 | 1 << 31 and the username MUST be “SERVER.” The message field MUST first contain the room name followed by “#RM#” followed by a space dilineated list of usernames for clients associated in that chat room.

If the room doesn’t exist, then the server creates a new message where the message_status value MUST be 1 << 12 | 1 << 31 and the username MUST be “SERVER.” The message field SHOULD be a string indicating that the room doesn’t exist.

4.6.2 Client parsing of server response

The client SHALL print the name of the room and each username, expecting that the room name is first in the message field followed by “#RM#” followed by user names that are each separated by a space.

4.7 Send a message to clients in chat rooms

The header message_status value is set to 1 << 6 and the username field MUST be set to reflect the client's chosen username. The message field MUST include each room the message is to be sent to with each room name separated by a '#' character. Room names and the client's message MUST be separated by "#RMS#" and the client's message MUST not exceed 1024 characters.

4.7.1 Server response

The server MUST first identify which rooms the client's message is to be forwarded to and expects the message content and room names to be separated by "#RMS#" characters. Furthermore, the server SHALL identify each individual room by parsing the the rooms expecting a '#' character to separate each room name. For each room name, the server SHALL first check if that room name exists. If it does exist, then the server MUST check if the originating client is in the room. If the originating client is in the room, then the server SHALL create a new message where the message_status MUST be set to 1 << 6 and the username field MUST be set to reflect the client's chosen username of the message's origin. The message field MUST first include the room name followed by "#RM#" followed by the message content. This message MUST be sent to all clients in the room, excluding the originating client.

If the room doesn't exist, then the server creates a new message where the message_status value MUST be 1 << 13 | 1 << 31 and the username MUST be "SERVER." The message field SHOULD be a string indicating that the room doesn't exist. This message MUST only be sent to the originating client.

If the room exists, but the originating client is not in the room, then the server creates a new message where the message_status value MUST be 1 << 13 | 1 << 31 and the username MUST be "SERVER." The message field SHOULD be a string indicating that the client is not in the room. This message MUST only be sent to the originating client.

4.7.2 Client parsing of server response

Upon receipt of a message with message_status set to 1 << 6, the client MUST print the room name, username, and the message content. The client SHALL expect that the message field contain the room name first followed by “#RM#” followed by the message content.

4.8 Send a private message to a client

The header message_status value is set to 1 << 7 and the username field MUST be set to reflect the client’s chosen username. The message field MUST include one username intended as the message’s destination followed by “#RMS#” followed by the message content. The client’s message content MUST not exceed 1024 characters.

4.8.1 Server response

The server MUST first identify which destination client the originating client’s message is to be forwarded to and expects the message content and destination client’s username to be separated by “#RMS#” characters. The server SHALL first check if that username exists. If the username does exist, then the server SHALL create a new message where the message_status MUST be set to 1 << 7 and the username field MUST be set to reflect the client’s chosen username of the message’s origin. The message field MUST only include message content. This message MUST be sent to all clients with a username that matches the supplied destination username, excluding the originating client.

If the destination client username doesn’t exist or it does, but only matches with the originating client’s username, then the server creates a new message where the message_status value MUST be 1 << 14 | 1 << 31 and the username MUST be “SERVER.” The message field SHOULD be a string indicating that the username doesn’t exist. This message MUST only be sent to the originating client.

4.8.2 Client parsing of server response

Upon receipt of a message with message_status set to 1 << 7, the client MUST in some way indicate that it is a private message and it MUST print the originating username, and the message content. The client SHALL expect that the message field contain only the message content.

4.9 Disconnect from server

The header message_status value is set to 1 << 8 and the username field MUST be set to reflect the client’s chosen username. The message field SHOULD be left empty.

4.9.1 Server response

The server SHALL first determine which chat rooms the client requesting a disconnect is associated with. The server SHALL then create a new message where the message_status field MUST be set to 1 << 8 and the username field set to the username of the client originating the disconnect. The message field SHALL include the room name and the username of the client originating the disconnect. This message MUST be sent to all clients in rooms that the client originating the disconnect was associated with, excluding the originating client. The server SHALL then remove the originating client from all rooms and terminate the connection with the client.

4.9.2 Client parsing server response

Upon receipt of a server response to a client disconnect, the client MUST print the message content.

5. Server Originated Messages

5.1 Server disconnect from clients

The username MUST be set to “SERVER” for these messages and the header message_status SHALL have a value of 1 << 7 | 1 << 31, indicating a message origin from the server. After sending the message(s), the server MUST terminate its connection with all clients and SHALL remove references to clients from any internal structures. The server SHALL NOT expect a response from clients.

5.1.1 Client response

The client MUST terminate its connection with the server.

6. Asynchronous I/O

Both the client and server will perform non-blocking reads from sockets that they are connected to and will also perform non-blocking reads from standard input. These non-blocking reads will allow the server to continue to attempt reads for the sockets of all connected clients as well as check for any standard input. Similarly, this allows the client to continue to receive input from a user without interrupting the ability to receive messages from the server. The server will maintain a list of sockets and input to read from that initially includes the socket that the server is listening for incoming connections on and standard input. As clients connect, the server will add these new connections to the list. As clients disconnect, their corresponding sockets will be removed from the list. For clients, they will only read from the socket connecting them to the server and standard input. Writes from the server and client may be blocking, but it is critical that read operations are non-blocking.

7. Error Handling

7.1 Client Error Handling

If the client detects that the server is no longer responding for any reason via the socket being broken, it must close its own connection on the socket. This will be handled by attempting to read from the socket. If any error is returned while attempting this, it must be assumed the socket, and consequently the connection to the server, is broken. If the client is unable to establish the initial connection with the server, it MAY continue to reattempt establishing the connection or simply terminate.

7.2 Server Error Handling

If the server detects a client is no longer responding for any reason via the socket connection being broken, it must close that connection and remove any notion that the client is in any chat room(s). Additionally, it must notify other clients that were in chat rooms with the unresponsive client that the unresponsive client has left the chat room. By attempting to read from the socket connection with a particular client, if the read returns an error for any reason, it must be assumed the socket, and thus the connection to that client, is broken.

8. Conclusion

Simple, generic internet relay chat application protocols have been specified that allow for simple communication between numerous clients that are connected to a single server. Additional work could easily incorporate additional mechanisms, including file transfers between clients. For the purposes of a better functioning private messaging feature, unique usernames could be enforced by the server. Future work could additionally remove the need for character and string delimiters, especially in the context of using JSON for message passing.