CS594
Group Name
Devyani Shrivastava
Harie Vashini Dhamodharasamy Kalpana

Internet Draft
Portland State University
Intended status: IRC Class Project Specification
December 01,2019
Expires: December 31, 2019

Internet Relay Chat Class Project
draft-irc-pdx-cs594-00.txt

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79. This document may not be modified, and derivative works of it may not be created, except to publish it as an RFC and to translate it into languages other than English.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on December 31, 2019.

Copyright Notice

Abstract

This memo describes the communication protocol for an IRC-style client/server system for the Internetworking Protocols class at Portland State University.

Table of Contents

1. Introduction

This specification describes a simple Internet Relay Chat (IRC) protocol by which clients can communicate with each other. This system employs a central server which ''relays'' messages that are sent to it to other connected users.

When IRC client logs into this application, users can create new “rooms” or join any “rooms” such that any message sent by any user in a room is broadcast to all other clients in that room. clients can join any number of existing rooms and leave any room. Users can also send a secure private message directly to other users. Users can also list the online clients. Server can transfer a file, requested by a user. At the end, User can quit the program, thus getting disconnected with server which will also remove the said user from all the groups that the user had joined.

2. Conventions used in this document

Values within the code piece presented in this document, and in the text of this document that are dynamically allocated or hard-coded will be enclosed within < >.

3. Basic Information

The Application rests on TCP, in a client-server architecture, with the server listening for connections on port 8080. The server is “always on”, and the client connects to the server by sending a “connection request” on the said port and maintains this persistent connection to the server. The client can send messages and requests to the server over this open channel asynchronously, and the server can reply via the same. The server may choose to allow only a finite number of users and rooms, depending on the implementation and resources of the host system. Error handling code is available at both ends; thus, a client will be gracefully informed if the server runs into any issue. Similarly, the server will also handle client-side issues gracefully.

4. Message Infrastructure

4.1. Generic Message Format

To enter command:
<command>
To send private message to another user:
private_chat <client_name> <message>

```
To send group message:
group_chat <name_of_the_room> <message>
Message received from person or group:
#from <user/group>: <message>
Message received from server:
     <server's message>
To receive file from the server:
     file_transfer
To broadcast a message:
     broadcast <message>
```

4.2. Error Message

If the command entered is not in the above-mentioned format, then the server returns an error message requesting the client to enter either the correct command or a different command.

5. Client Messages

5.1 First message sent to server

As soon as the client is run, it is prompted to enter a username. This username is sent to the server and the server makes sure if the entered name is unique. If it is not unique, the client is requested to enter a new name. The process will be repeated until the client enters a unique name or quits the application. After entering a unique name, the client enters password and then the client is said to be connected.

5.1.1. Usage

Once the connection has been established, and before any other communication can occur, the client must provide a unique username.

The server associates this name with the client's socket connection, using a hash-map whose key is the name of the client and value is the socket connection of that client.

5.2 List online clients

In order to view all clients who are online, the user is required to enter the below command:
list_clients

Server returns the keys of the hashmap clients where it has maintained the list of client socket connections currently alive.

5.3 Listing Room

If client wants to see the list of existing rooms/groups, the user is required to enter the command “list_rooms”.

5.4 Listing members in a room

If client wants to see the list of all existing users in a room, the user is required to enter the command:
“list_members <name_of_the_room>”

The server responds with a list of existing members of that room. Server maintains a hashmap whose key is name of the room, and value is the list of clients registered to that room.

5.5 Creating Room

To create a room: create_room <name_of_the_room>

The above command is entered by the client to create a chat room. If no room by that name exists, one is created. If a room is already available with the same name, the server responds “sorry, Room name already available. Enter another.”

The server registers the new room along with the client that created it, by default.

5.6 Joining Room

To join the room: join_room <name_of_the_room>

The above command is sent by the client to join any existing room. The server adds the user to that room if the room exists and if the user is not a member in it, else it sends a message “already in the room” if the user is already a member in that room. Server sends a message “This room does not exist”, if there is no room in that name.

5.7 Leaving Room

To leave a room: quit_room <name_of_the_room>

The above command is sent by the client to leave a chat room.

The server removes the client’s registration from that room in response to the message otherwise it sends “not in this room, enter a valid room” if the user is not a member of that room.

5.8 Private chat

To send personal messages to other users, users are required to enter

Private_chat <client_name> <message>

Private messages sent by one client to another client are visible only to the intended recipient client. This is a secure transfer.

Server receives the message, parses the string to obtain the name of the other client, and sends the encrypted message to the other client. The client then decrypts the message to obtain the original message.

5.9 Group chat

To send a message to a room/group, user should enter the following command:

group_chat <name_of_the_room> <message>

Sent by client to all the members logged in to the room.

Server iterates over the value of the hashmap where key is the name of the group and sends a message to all connected clients.

6. Server Messages

6.1 Listing Responses

Two types of hashmaps maintained by the server: one is to record all the online clients, two is to record existing room names and their registered clients. Hashmap values are converted to string and sent.

client.send(<message>.encode())

6.2 Forwarding Messages

Messages received from a client is parsed to investigate the purpose of the message. If the message is meant for another client, the message is passed to that client via its socket connection.

If the message is meant to be sent to a room, the server iterates over the list of members in the room and forwards the message to each client via their individual socket connection.

clientusers[i].send(<message>.encode())

Here clientusers is a dictionary that consists of all the online clients. ‘i’can refer to members in a room or an individual client based on the commands (private_chat or group_chat).

7. Error Handling

If the server detects the loss of client connection, the server MUST remove the client from all rooms to which they belong. If the server disconnects, all the clients receive a prompt that the server has been disconnected and they should shut the connection. They may choose to connect to the server again, if the server is up.

8. “Extra” Features Supported

8.1 Secure Private Messaging

Note that private messaging is supported in addition to meeting the other remaining project criteria. Private chat allows the client to transfer a secure message to the other client. The message sent is encrypted by the server and in the recipient side, the client decrypts the message to get the original message. Thus, server provides a secure message transfer. Here the clients use Caesar cipher encryption, decryption to perform a secure transmission.

8.2 Broadcast

The client can send a common message to all online clients. The format of the message is similar to the existing format.
To broadcast a message, we use:
broadcast <message>

8.3 Receive file from the server

The client sends the command “file_transfer” to the server in order to receive the file. The server accesses the “source.txt” file, which resides in the directory of the server program. It opens and reads the file and sends the file line by line to the client. The client writes the received content in “destination.txt” file.

9. Conclusion

This specification provides a generic message passing framework for multiple clients to communicate with each other via a central forwarding server.

Without any modifications to this specification, it is possible for clients to devise their own protocols that rely on the text-passing system described here. This specification is the first draft of the IRC Project hence-forth undertaken, which considers the transport layer protocols and the services provided by it. Based on which a plan of the application has been created, a framework through which multiple clients can communicate with each other via a server.

10. Security Considerations

Messages sent using this system have no protection against inspection, tampering or outright forgery. The server can see all the messages that are sent through it. “private messaging” is an exception. clients implement a cryptographic algorithm named Caesar cipher to provide a secure communication for private messages.

11. Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997

12. Acknowledgements

This document was prepared using Microsoft Word.