Internet Draft

Matt Geary
Jason Brewer
June 2019

CS594 Internet Relay Chat Protocol

Abstract

This document describes the communication protocol of the CS594 Internet Relay Chat system.

Contents

1 Introduction

The CS594 IRC (Internet Relay Chat) protocol described herein is based on a client/server model which allows multiple clients to connect to a single server. Server-to-server connections are not considered. The central server relays messages sent to it by users.

Users are able to join rooms. A room serves as a message stream which users can subscribe/unsubscribe to by joining or leaving. The server tracks which clients are in which rooms, while each client knows which room it is in. The server is able to provide clients with a list of rooms when requested, as well as providing clients with a list of members of each room.

Any client can send/receive a private message to/from any other client.

1.1 Server

The server is the central controller mechanism which functions as the sole point of connectivity to the client. The server will manage connection information and will send messages to clients. The server is multithreaded; one server thread waits for new client connections and, as each connection is received, that connection is given its own socket and thread of control with which to interact with that client. The server can be set up to use any port for connections. The client-server messaging protocol is asynchronous; the client can send messages at any time and the server can send messages at any time. The client and server are each permitted to close the connection at any time, with or without informing the other.

1.2 Client

The client is an application that connects to the server. Clients are recognized by a unique screenname and set of connection information, all of which is stored by the server. A client will connect to a server using a Windows or Linux socket; once a connection has been established, the client can then interact with the server by sending specific message protocols described herein.

2 Message Details

Messages are sent between client and server and vice versa. ALL messages MUST be sent in UTF-8 encoding. Server messages are sent based on valid inputs from the client.

All messages consist of up to four main parts: a command code, code parameters, message marker and a message. The command must be a valid IRC command as described in this protocol. Unknown commands will result in a server response of "Verified". This server response

signifies that the server received and parsed the message, but did not detect any action to take. The command parameters, message marker and message may be optional depending on the command. All parts of the message are delineated by means of a space (UTF-8 0x20) character. A message cannot be longer than 4096 bytes. The client and server must both enforce this limit; if the message is longer, the client and server can either choose to reject it and throw an error, or else send the message cut down to 4096 bytes.

2.1 Client Operations

This section deals with client-side functionality and the specific message protocols the IRC can utilize.

2.1.1 User Names

After connecting to the server, the FIRST message a client sends to the server will be interpreted by the server as their desired screenname. The server will then check this name against a list of in-use names, and if the name is in use, the server will send back “NAMEERROR" and close the connection. The client must terminate or prompt the user for a new username upon receiving the “NAMEERROR" message. Screen names can be no more than twenty characters long. There is currently no way to change the screen name after initial selection. When a client disconnects from the server, their name is removed from the in-use list, permitting it to be used by another connection.

2.1.2 Creating a room

```
code: new
parameters: [name] ([name] ...)
```

To create a room, the client must send the message “new", followed by a sequence of names delineated by a space (0x20). Clients can create multiple rooms at once by means of specifying multiple room names. Each room name must be unique, otherwise the server will message the client that the room name is already in use and active. If the client sends a list of names of which some are valid and some invalid, the server will create the valid rooms while rejecting the invalid rooms. If no names are supplied, the server will inform the user of missing information.

2.1.3 Displaying rooms

```
code: rooms
```

The client sends the message “rooms" to the server; if there are rooms on the server, the server will send the client a list of all the active rooms. When there are no rooms, the server will inform

the client that there are no active rooms.

2.1.4 Joining a room

```
code: join
parameters: [name] ([name] ...)
```

To join a specified room, the client must send the message “join" followed by a sequence of room names delineated by a space (0x20). The IRC allows a client to join multiple rooms at the same time. However, the rooms specified by the client must be valid room names. If any of the names are invalid, the server will inform the client and no rooms will be joined.

2.1.5 Leaving a room

```
code: leave
parameters: [name] ([name] ...)
```

Sending message “leave" allows the client to leave multiple rooms at once. The client must specify valid room names, delineated by a space (0x20). If some of the room names are valid and others are not, the client will only be removed from the valid rooms.

2.1.6 Displaying members in a room

```
code: members
parameters: [name]
```

Sending message “members" will prompt the server to return to the client a list of everyone in the specified room. If there are no people in the room, the server will send a message stating that there are no members. The client must supply a valid room name. The client can only query membership of one room at a time.

2.1.7 Sending a message to room

```
code: send
parameters: [name] ([name] ...)
message marker: -
message: [string]
```

A “send" message will send a string to selected rooms. A client must specify at least one valid room name to send a message to. The message string can be any combination of UTF-8 characters and will be displayed to all users in the specified room(s), including the sender.

2.1.8 Sending a message to another client

```
code: tell
parameters: [name]
message: [string]
```

A “tell" message will send a message to the client associated with the specified username. The tell command takes a user name as a parameter. This user name must be valid, otherwise the server will send a response back to the client. Upon success, a message will be sent to both the sender and the receiver.

2.1.9 Exiting

```
code: exit
```

An “exit" message will inform the server of the client exiting. Upon receipt, the server will terminate the client's connection and remove the client's username from the in-use list.

2.2 Server Operations

2.2.1 User Name

```
code: NAMEERROR
```

The FIRST message received from a client will be interpreted by the server as their desired screenname. The server will then check this name against a list of in-use names, and if the name is in use, the server will send back message “NAMEERROR" and close the connection. Otherwise, the server must append the client name to a list of in-use names.

2.2.2 Room Join

```
code: Joined
message: [roomlist]
```

After receiving and processing a “join" message from the client, the server will send message “Joined" followed by a space (0x20) followed by a list of the rooms the client tried to join, alongside notifiers of whether each join succeeded or failed. Format of the message/notifiers is not specified by this protocol.

2.2.3 Room Removal

```
code: Left
```

After receiving and processing a “leave" message from the client, the server will send message “Left" followed by a space (0x20) followed by a list of the rooms the client tried to leave, alongside notifiers of whether each leave succeeded or failed. Format of the message/notifiers is not specified by this protocol.

2.2.4 Message Relay

```
rooms_message: [user] says in room [name] [text]
private_message_send: You tell [user] [text]
private_message_rcv: [user] whispers to you [text]
```

```
After receiving and processing a ‘‘send" message from the client, the serve
After receiving and processing a ‘‘whisper" message from the client, the se
```

2.2.5 Room List

```
message: [roomlist]
```

After receiving a “rooms" message, the server will send the client a list of the rooms on the server. Format of the message/notifiers is not specified by this protocol.

2.2.6 Member List

```
message: [memberlist]
```

After receiving a “members [name]" message, the server will send the client a list of the members of room [name]. Format of the message/notifiers is not specified by this protocol.

```
message: [memberlist]
```

After receiving a “members [name]" message, the server will send the client a list of the members of room [name]. Format of the message/notifiers is not specified by this protocol.

2.2.7 Message Relay

After receiving and processing a

```
message: [memberlist]
```

2.3 Other Operations

These operations are intended to be used in an administrative capacity but are currently accessible for any client.

2.3.1 Displaying clients

```
code: clients
```

A “clients" message will display the user names of all clients that the server is currently maintaining.

2.3.2 Killing a server

```
code: kill
```

A “kill" message will inform the server to shut itself down. Upon receipt, the server will close all sockets to all client(s), and will then exit. The remaining client(s) will be disconnected and their interface will terminate.

3 Error Handling

3.0.1 Unexpected Client Termination

If the client terminates unexpectedly for any reason, it cannot be expected to send a message to the server. The server thread for that client MUST be able to detect the terminated connection. This can be easily accomplished by having the listening socket catch the resulting exception, closing the socket and terminating the thread. Windows Errors 10038, 10053, 10054 indicate a terminated connection. On linux, err 9 can be caught to indicate a terminated connection.

3.0.2 Unexpected Server Termination

If the server terminates unexpectedly for any reason, it cannot be expected to send a message to the client(s). The client MUST be able to detect the terminated connection. This can be easily accomplished by having the listening socket catch the resulting exception, closing the socket and terminating the client application, or else offering an option to attempt reconnection. Windows Errors 10038, 10053, 10054 indicate a terminated connection. On linux, err 9 can be caught to signify a terminated connection.

4 Security Considerations

Currently, the IRC presented has no security features. All clients are accessible via the server and are thus subject to any tampering. Rooms are not password protected, so a client can join any room. Additionally, messages are not encrypted when transmitted.

5 Future Work

There are number of features that are not currently implemented in the CS594 IRC system. Further work would be concentrated on three key areas:

- Client privileges: There is currently no privilege separation on the client-side between an administrator and a normal user. For instance, all clients have the ability to shut down the server. Future work would have to create super users with advanced privileges.
- Client-side security : Client accounts do not require passwords to connect. It would be beneficial to save client information on the server side and put them in an inactive status, and then activate the client on return once the client has input the correct password. This would save state, and would allow for more functionality on the client side (e.g. a friends list, auto-join rooms, etc.)
- Server-side security : Messages that are routed via the server possess no encryption. It would be beneficial for clients to send public keys along with the message to allow the server to route secure messaging between clients. Additionally, information kept on the server is easily subject to misuse. Lists of client records are maintained openly via the server. It would be beneficial to keep passwords encrypted on the server-side and not leave it open to inspection.

6 Acknowledgements

This document was prepared using the OGF Document Template. Further information can be found at `https://www.overleaf.com/latex/templates/ogf-document-template/wtghstbyrgny`.