CS594 Internet Draft
Intended status: IRC Class Project Specification
Expires: November 2023

Amelia Miner, Sarvesh Biradar
Portland State University

May 21, 2023

Internet Relay Chat Class Project

Status of this Memo

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 document specifies a basic Internet Relay Chat (IRC) protocol whose purpose is to facilitate communication between clients across the internet. Client applications send messages to a central server, which relays the messages to other connected users.

Clients can join rooms, which act as message streams to which multiple users can subscribe. Any message sent to a room is relayed to all clients who have joined that room. Clients are also able to leave rooms, effectively unsubscribing them from the stream of messages.

2. Conventions Used In This Document

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lowercase uses of these words are not to be interpreted as carrying significance described in RFC 2119.

3. Basic Information

This protocol is intended to run on top of TCP/IP. The server listens for client connections on port 7734. Each Client communicates with the server over a persistent connection on this port. Both inbound and outbound communication happens over this open connection. In this manner communication between the server and the client is inherently asynchronous.

Either server or client may terminate the connection at any time and for any reason. Either MAY send an error message to the other including a reason for termination of the connection.

The server MAY have a maximum number of allowed users in order to operate under implementation or resource constraints. The server MAY respond to connection requests in excess of this maximum with an appropriate error message. For more details on error messages see [4.2].

4. Message Infrastructure

4.1. Generic Messages

```
struct irc_header {
     uint8_t opcode;
     uint32_t len;
};

struct irc_pkt_msg {
     struct irc_header header;
     char payload[header.len];
};
```

4.1.1. Field definitions

- header.opcode - opcode specifies the purpose of the message; what type of message is contained in the payload, if any.

- header.len - len specifies the length of the payload in bytes, not including the header itself.

- Client and server must both check that the 'len' field is valid for the associated opcode. If the len is not within valid range for the opcode, the entity that detects the discrepancy MUST terminate the connection, and it MAY provide the appropriate error code to the opposite party (see [4.2]).

- payload - variable-length payload used by some but not all message types.

4.1.2. Operation Codes (opcodes)

IRC_ERR	= 0x00
IRC_KEEPALIVE	= 0x01

IRC_HELLO	= 0x02
IRC_LISTROOMS	= 0x03
IRC_LISTUSERS	= 0x04
IRC_JOINROOM	= 0x05
IRC_LEAVEROOM	= 0x06
IRC_SENDMSG	= 0x07
IRC_LISTROOMS_RESP	= 0x08
IRC_LISTUSERS_RESP	= 0x09
IRC_TELLMSG	= 0x0A

4.2. Error Messages

```
struct irc_pkt_err {
     struct irc_header header = {
          .opcode = IRC_ERR;
          .len = 1;
     };
     uint8_t err_code;
};
```

4.2.1. Usage

An irc_pkt_err packet MAY be sent by either a client or server prior to closing their TCP connection. The packet serves to inform the other party of the reason for closing the connection. If either party receives this message, that entity SHOULD consider the connection terminated.

4.2.2. Field definitions

- err_code - specifies the type of the error that warranted terminating the connection.

4.2.3. Error Codes

IRC_ERR_UNKNOWN	= 0x10
IRC_ERR_ILLEGAL_OPCODE	= 0x11
IRC_ERR_ILLEGAL_LEN	= 0x12
IRC_ERR_ILLEGAL_LABEL	= 0x13
IRC_ERR_ILLEGAL_MESSAGE	= 0x14
IRC_ERR_WRONG_VER	= 0x15
IRC_ERR_NAME_EXISTS	= 0x16
IRC_ERR_TOO_MANY_USERS	= 0x17
IRC_ERR_TOO_MANY_ROOMS	= 0x18

4.3. Keepalive Messages

```
struct irc_pkt_keepalive {
     struct irc_header header = {
          .opcode = IRC_KEEPALIVE;
          .len = 0;
     };
};
```

4.3.1. Usage

keepalive messages MUST be sent at least once every 5 seconds by both the client and the server to maintain each open connection.

Both client and server SHOULD consider the other party disconnected if more than some finite amount of time passes without receipt of a keepalive message. This timeout period MUST exceed 15 seconds if it is set.

5. Label Semantics

All users and rooms have identifiers, which MUST be validated as follows:

- Field length is fixed at 32 bytes.
- Must consist of human-readable ASCII character values between 0x20 and 0x7E.
- Must be at least 1 character and at most 32 characters in length.
- Cannot start or end with a space.
- If the label is less than 32 characters in length it must be null-terminated (the final character must be followed by 0x00). Remaining characters MAY also be null.
- If any of these checks fail at any point, the entity that detects the failure MUST terminate the connection and MAY provide the appropriate error (see [4.2]).
- Before using the data extracted from this field, the entity SHOULD append a null character to reduce likelihood of buffer overflow attacks.

6. Client Messages

6.1. First Message Sent to the Server

```
struct irc_pkt_hello {
     struct irc_header = {
          .opcode = IRC_HELLO;
          .len = 34;
     };
     uint16_t ver = 0x1337;
     char user_name[32];
};
```

6.1.1. Usage

Once connected, the client MUST send a hello packet including a valid name label and protocol version number.

The server MUST associate the client's user_name with the connection when the user_name is available. In the case that the client requests a name that is unavailable, the server MUST terminate the connection with the appropriate error message (IRC_NAME_EXISTS). The hello message SHOULD only be sent once for each connection by each client; if the server receives further hello messages, the server MAY ignore them or terminate the client's connection.

6.1.2. Field Definitions

- ver - used to determine whether the client is using the same version of the protocol as the server. If the server receives a value that does not match the expected value of 0x1337, the server MAY alert the client using the appropriate error code and MUST terminate the connection.
- user_name - specifies the name label requested by the client. MUST follow label semantics as defined in [5].

6.2. Listing Rooms

```
struct irc_pkt_list_rooms {
     struct irc_header header = {
          .opcode = IRC_LISTROOMS;
          .len = 0;
     };
};
```

6.2.1. Usage

Clients can send this message to request a list of all of the rooms currently occupied by at least one client.

6.2.2. Response

The server MUST return a message with the IRC_LISTROOMS_RESP opcode. The body of this message MUST contain a list of all names of currently-occupied rooms. See [7] for details.

6.3 Listing Users

```
struct irc_pkt_list_users {
     struct irc_header header = {
          .opcode = IRC_LISTUSERS;
          .len = 32;
     };
     char room_name[32];
};
```

6.3.1. Usage

Clients can send this message to request a list of all of the users who are joined to the specified room.

6.3.2. Response

The server MUST return a message with the IRC_LISTUSERS_RESP opcode. The body of this message MUST contain a list of all names of users in the requested room if any. See [7] for details.

6.4 Joining and Creating Rooms

```
struct irc_pkt_join_room {
     struct irc_header header = {
          .opcode = IRC_JOINROOM;
          .len = 32;
     };
     char room_name[32];
```

```
    };
```

6.4.1. Usage

Clients can send this message to join a room. If no such room exists and the server is not prevented from creating a new room due to resource or implementation constraints, the server MUST create a new room with the requested room_name.

When a user joins a room, the server MUST send a message with the IRC_LIST_USERS_RESP opcode to all users currently in that room, and the body of this message MUST contain a list of all users present in the room. The identifier field MUST contain the room name, and the item_names list MUST contain all users in the room. See [7] for details.

6.4.2. Field Definitions

- room_name - The name of the room to join or create. MUST follow label semantics (see [5]).

6.5. Leaving a Room

```
struct irc_pkt_leave_room {
     struct irc_pkt_header header = {
               .opcode = IRC_LEAVE_ROOM,
               .len = 32
          };
          char room_name[32];
}
```

6.5.1. Usage

The client can send a message to indicate their intention to exit a room.

When the server receives this message, it MUST remove the client from the designated room and MUST send an "irc_pkt_list_users" message to all the users who are still in that room, as there has been a change in the user list. The "identifier" field in the "list_users" message MUST be set to the room name, and the "item_names" list MUST contain the names of all users present in the room.

The server SHOULD disregard leave requests from clients who are not currently members of the specified room.

6.5.2. Field Definitions

- room_name - The name of the room to join or create. MUST follow label semantics (see [5]).

6.6. Sending Messages

```
struct irc_pkt_send_msg {
     struct irc_pkt_header header = {
          .opcode = IRC_SEND_MSG <OR> IRC_SEND_PRIV_MSG,
          .len = LENGTH
     };
     char target_name[32];
     char msg[LENGTH - 32];
}
```

6.6.1. Usage

A client can use this command to send a text message either to a channel room or to another user.

If the messages' opcode is IRC_SEND_MSG, it indicates that the message is intended for a room. Once the server validates this message, it MUST send an IRC_TELL_MSG message to all users present in the specified room. In the IRC_TELL_MSG message, the "target_name" parameter should be set to the room name, and the "sending_user" parameter should be set to the name of the user who sent the message. It's worth noting that the server MAY send messages to rooms even if the client is not a member of those rooms.

On the other hand, if the opcode is IRC_SEND_PRIV_MSG, it signifies that the message is meant for another user. After validating this message, the server MUST send an IRC_TELL_PRIV_MSG message to the designated user. The forwarded message should be included in the IRC_TELL_PRIV_MSG message, and the "sending_user" parameter should be set to the name of the user who initially sent the message. The "target_name" value in the response is not used.

6.6.2. Field Definitions

- target_name - Name of the entity to send the message to.
- msg - Message to send to the user.
  MUST be less than 8000 characters long.
  MUST consist entirely of readable ASCII character values, between 0x20 and 0x7E, and additionally can contain the carriage return and newline characters (0x0D and 0x0A).
  MUST be NULL terminated.
  MUST not contain extra NULL terminators within the payload, they may only be at the very end. If the message breaks any of these rules, the server MUST terminate the connection, and MAY provide the error code IRC_ERR_ILLEGAL_MESSAGE.

7. Server Messages

7.1. Listing Response

```
struct irc_pkt_list_resp {
     struct irc_pkt_header header = {
          .opcode = IRC_LIST_ROOMS_RESP <OR> IRC_LIST_USERS_RESP,
          .len = LENGTH
     };
     char identifier[32];
     char item_names[LENGTH/32 - 1][32];
}
```

7.1.1. Usage

The server sends a generic response message to the client, providing information about a list. This message is used for both listing rooms and listing users within a room.

7.1.2. Field Definitions

- identifier - Used only for IRC_LIST_USERS_RESP, contains the name of the room to which the users belong. MUST follow label semantics.

- item_names - Array of item names, MUST follow label semantics.

7.2. Forwarding Messages to Clients

```
struct irc_pkt_tell_msg {
     struct irc_pkt_header header = {
          .opcode = IRC_TELL_MSG <OR> IRC_TELL_PRIV_MSG,
          .len = LENGTH
     };
     char target_name[32];
     char sending_user[32];
     char msg[LENGTH - 64];
}
```

7.2.1. Usage

In the case of the opcode being IRC_TELL_MSG, it denotes a message forwarded by the server to indicate that the specified message has been posted to a room the client has joined. The server MUST set the name of the room to the "target_name" field. Clients SHOULD disregard this message if the "target_name" does not match any of the rooms the client believes it is a member of.

Alternatively, if the opcode is IRC_TELL_PRIV_MSG, it signifies a forwarded private message sent by another user intended specifically for the recipient. The server MUST assign the name of the intended recipient to the "target_name" field. Clients MAY verify if this value is correct.

7.2.2. Field Definitions

- target_name - Name of the user or room that the message was sent to.
- sending_user - Name of the user who sent the message
- msg - Message sent to the room.

MUST be less than 8000 characters long
MUST consist entirely of readable ASCII character values, between 0x20 and 0x7E, and additionally can contain the carriage return and newline characters (0x0D and 0x0A).

8. Error Handling

Both the server and the client MUST identify when the socket connection between them is terminated. This can be achieved by actively sending the traffic or by keeping track of heartbeat messages. In the event that the server detects the loss of the client's connection, it MUST remove the client from all the rooms they have joined. On the other hand, if the client detects that the connection to the server has been lost, it MUST recognize that it is disconnected and MAY decide to initiate a reconnection process.

As mentioned earlier, there is the possibility of one party choosing not to inform the other party about an error occurrence.

9. “Extra” Features Supported

TBD

10. Conclusion & Future Work

The purpose of this specification is to establish a generic framework for message passing, enabling multiple clients to communicate with one another through a central forwarding server.

This specification allows clients to create their own protocols utilizing the text-passing system described here, without requiring any modifications. One example is the transfer of binary data, which can be accomplished by encoding it in base64 format. This infrastructure can facilitate the transmission of files of any size or enable the establishment of secure connections by utilizing cryptographic transport protocols like Transport Layer Security (TLS).

11. Security Considerations

The messages transmitted through this system lack safeguards against inspection, tampering, or forgery. The server has visibility into all the messages exchanged via this service. Consequently, so-called "private" messages can be intercepted by third parties with the capability to capture network traffic and their contents are visible to the server being used to relay them.

12. IANA Considerations

None.

12.1. Normative References

TBD

13. Acknowledgments

This document was prepared using Google Docs.