CS594
Internet Draft Portland State University
Intended status: IRC Class Project Specification December 1, 2015
Expires: June 2016

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

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 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.

Users can join rooms, which are groups of users that are subscribed to the same message stream. Any message sent to that room is forwarded to all users currently joined to that room.

Users can also send private messages directly to other users.

2. Conventions used in this document

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 RFC 2119 [RFC2119].

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

In this document, the characters ">>" preceding an indented line(s) indicates a statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the portions of this RFC covered by these keywords.

3. Basic Information

All communication described in this protocol takes place over TCP/IP, with the server listening for connections on port 7734. Clients connect to this port and maintain this persistent connection to the server. The client can send messages and requests to the server over this open channel, and the server can reply via the same. This messaging protocol is inherently asynchronous - the client is free to

send messages to the server at any time, and the server may asynchronously send messages back to the client.

As is described in [4.2], both the server and client may terminate the connection at any time for any reason. They MAY choose to send an error message to the other party informing them of the reason for connection termination.

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 codes are available to notify connecting clients that there is currently a high volume of users or groups accessing the server.

4. Message Infrastructure

4.1. Generic Message Format

```
struct irc_pkt_header  {
    uint32_t opcode;
    uint32_t length;
};

struct irc_pkt_generic {
    struct irc_pkt_header header;
    uint8_t payload[header.length];
};
```

4.1.1. Field definitions:

- header.opcode - specifies what kind of message is contained within the payload.
- header.length - specifies how many bytes of payload follow the header in this message (excludes header size).
- Both client and server MUST validate that the length is valid for the provided opcode. If not, the entity that detects the error MUST terminate the connection, and MAY provide the error code IRC_ERR_ILLEGAL_LENGTH to the opposite party (see error messages section).
- payload - variable length payload. Not used by some messages.

4.1.2. Operation Codes (opcodes)

```
IRC_OPCODE_ERR             = 0x10000001
IRC_OPCODE_KEEPALIVE       = 0x10000002
IRC_OPCODE_HELLO           = 0x10000003
IRC_OPCODE_LIST_ROOMS      = 0x10000004
IRC_OPCODE_LIST_ROOMS_RESP = 0x10000005
IRC_OPCODE_LIST_USERS_RESP = 0x10000006
IRC_OPCODE_JOIN_ROOM       = 0x10000007
IRC_OPCODE_LEAVE_ROOM      = 0x10000008
IRC_OPCODE_SEND_MSG        = 0x10000009
IRC_OPCODE_TELL_MSG        = 0x10000010
IRC_OPCODE_SEND_PRIV_MSG   = 0x10000011
IRC_OPCODE_TELL_PRIV_MSG   = 0x10000012
```

4.2. Error Messages

```
struct irc_pkt_error {
    struct irc_pkt_header header =
        { .opcode = IRC_OPCODE_ERR, length = 4 };
    uint32_t error_code;
}
```

4.2.1. Usage

MAY be sent by either the client or the server prior to closing the TCP connection to inform the other party why the connection is being closed. If either client or server receives this message, that entity SHOULD consider the connection as terminated.

4.2.2. Field definitions:

- error_code - specifies the type of error that occurred

4.2.3. Error Codes

```
IRC_ERR_UNKNOWN         = 0x20000001
IRC_ERR_ILLEGAL_OPCODE  = 0x20000002
IRC_ERR_ILLEGAL_LENGTH  = 0x20000003
IRC_ERR_WRONG_VERSION   = 0x20000004
IRC_ERR_NAME_EXISTS     = 0x20000005
IRC_ERR_ILLEGAL_NAME    = 0x20000006
IRC_ERR_ILLEGAL_MESSAGE = 0x20000007
IRC_ERR_TOO_MANY_USERS  = 0x20000008
IRC_ERR_TOO_MANY_ROOMS  = 0x20000009
```

4.3. Keepalive Messages

```
struct irc_pkt_keepalive {
    struct irc_pkt_header header =
        { .opcode = IRC_OPCODE_ERR, .length = 0 };
}
```

4.3.1. Usage

Provides an application-layer notification of a disconnected peer.

MUST be sent at least once every 5 seconds by both client and server to notify the other party that they are still connected.

Both client and server SHOULD watch for these keepalive messages and consider the other party disconnected if more than some set period of time has elapsed. If such a timeout is used, the time MUST exceed 15 seconds.

5. Label Semantics

Identifying both users and rooms involves sending and receiving labels. All label rules are the same, and MUST be validated as follows:

- Field size for transmission is always exactly 20 bytes.
- Must consist entirely of readable ASCII character values, between 0x20 and 0x7E.
- Must be at least 1 character, and at most 20 characters.
- Cannot start or end with a space.
- If less than 20 characters, the first character following the message MUST be a null character (0x00). Remaining characters MAY also be null characters.
- If any of these rules are broken, the receiver MUST terminate the connection and MAY provide the IRC_ERR_ILLEGAL_NAME error.
- Before using this field, recipient SHOULD append a NULL terminator to this array to reduce the likelihood of a buffer overflow attack.

6. Client Messages

6.1. First message sent to the server

```
struct irc_pkt_hello {
    struct irc_pkt_header header =
        { .opcode = IRC_OPCODE_HELLO, .length = 24 };
    uint32_t ver_magic = 0xFACE0FF1;
    char chat_name[20];
};
```

6.1.1. Usage

Before subsequent messages can be sent, a connecting client MUST provide a chat name and identify which version of the protocol they are using.

The server MUST associate the client's chat_name with the socket connection of the user. This message SHOULD be sent only once; if the server receives the message more than once, the server MAY ether ignore the message or terminate the client's connection.

6.1.2. Field Definitions

- ver_magic is used to determine if the client is using the same version of the protocol. If the server receives any other value than the expected value 0xFACE0FF1, the server MAY alert the client using the error code IRC_ERR_WRONG_VERSION, and MUST terminate the connection.

- chat_name specifies the name the connecting client wishes to use. MUST follow label semantics.

- chat_name must not be the same name provided by any other currently connected client. If the name already exists, the server MUST return the error IRC_ERR_NAME_EXISTS. The client can then attempt to reconnect with a different name.

6.2. Listing Rooms

```
struct irc_pkt_list_rooms {
    struct irc_pkt_header header =
        { .opcode = IRC_OPCODE_LIST_ROOMS, .length = 0 };
}
```

6.2.1. Usage

Sent by the client to request a list of all of the rooms currently occupied by at least one other client.

6.2.2. Response

Server MUST return an irc_pkt_list_resp with opcode IRC_OPCODE_LIST_ROOMS_RESP with a list of the names of all currently occupied rooms.

6.3. Joining and Creating Rooms

```
struct irc_pkt_join_room {
    struct irc_pkt_header header =
        { .opcode = IRC_OPCODE_JOIN_ROOM, .length = 20 };
    char room_name[20];
}
```

6.3.1. Usage

Sent by the client to join a chat room. If no room by that name exists, one is created.

Upon joining a room, the server MUST send an IRC_OPCODE_LIST_USERS_RESP message to all users currently in that room to alert them that the user list has changed. The identifier MUST be set to the name of the room, and the item_names list MUST include a list of all of the users in the room.

Every time the room's user list changes the server MUST send a new IRC_OPCODE_LIST_USERS_RESP message to all users in the room informing them of the new room membership.

6.3.2. Field Definitions

- room_name - Name of the room to join or create. MUST follow label semantics.

6.4. Leaving a Room

```
struct irc_pkt_leave_room {
    struct irc_pkt_header header =
        { .opcode = IRC_OPCODE_LEAVE_ROOM, .length = 20 };
    char room_name[20];
}
```

6.4.1. Usage

Sent by the client to leave a chat room.

Upon receiving this message the server MUST remove the client from the specified room and MUST send an IRC_OPCODE_LIST_USERS_RESP message to all users currently in that room to alert them that the user list has changed. The identifier MUST be set to the name of the room, and the item_names list MUST include a list of all of the users in the room.

The server SHOULD ignore leave requests when the client is not currently a member of the specified room.

6.4.2. Field Definitions

- room_name - Name of the room to leave. MUST follow label semantics.

6.5. Sending Messages

```
struct irc_pkt_send_msg {
    struct irc_pkt_header header =
        { .opcode = IRC_OPCODE_SEND_MSG
              <OR>  IRC_OPCODE_SEND_PRIV_MSG, .length = LENGTH };
    char target_name[20];
    char msg[LENGTH - 20];
}
```

6.5.1. Usage

Sent by a client to send a text message to either a room or another user.

If the opcode is IRC_OPCODE_SEND_MSG, then the target is a room and, after validating this message, the server MUST send an IRC_OPCODE_TELL_MSG message to all users in the specified room with the target_name parameter set to the room name, and the sending_user parameter set to the name of the user who sent the message. The server MAY choose to send messages to rooms that the client is not a member of.

If the opcode is IRC_OPCODE_SEND_PRIV_MSG, then the target is another user and, after validating this message, the server MUST send an IRC_OPCODE_TELL_PRIV_MSG message to that specific user with the forwarded message and the sending_user parameter set to the name of

the user who sent the message. The target_name value in the response is unused.

6.5.2. Field Definitions

- target_name - Name of the entity to send the message to.

- msg - Message to send 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).

  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_OPCODE_LIST_ROOMS_RESP
             <OR>   IRC_OPCODE_LIST_USERS_RESP, .length = LENGTH };
    char identifier[20];
    char item_names[LENGTH/20 - 1][20];
}
```

7.1.1. Usage

Generic listing response message sent by the server to inform the client of a list. Used for both listing rooms and listing users in a room.

7.1.2. Field definitions

- identifier - Used only for IRC_OPCODE_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_OPCODE_TELL_MSG
              <OR>  IRC_OPCODE_TELL_PRIV_MSG, .length = LENGTH };
    char target_name[20];
    char sending_user[20];
    char msg[LENGTH - 40];
}
```

7.2.1. Usage

If the opcode is IRC_OPCODE_TELL_MSG, then this is a forwarded message from the server indicating that the specified msg was posted to a room the client is joined to. Server MUST set the target_name field to the name of the room to which the message belongs. Clients SHOULD ignore this message if target_name does not match one of the rooms the client believes it is joined to.

If the opcode is IRC_OPCODE_TELL_PRIV_MSG, then this is a forwarded private message from another user intended specifically for the recipient. Server MUST set the target_name field to the name of the intended recipient. Clients MAY choose to confirm 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 server and client MUST detect when the socket connection linking them is terminated, either when actively sending traffic or by keeping track of the heartbeat messages. If the server detects that the client connection has been lost, the server MUST remove the client from all rooms to which they are joined. If the client detects that the connection to the server has been lost, it MUST consider itself disconnected and MAY choose to reconnect.

As stated previously, it is optional for one party to notify the other in the event of an error.

9. "Extra" Features Supported

Note that private messaging is supported in addition to meeting the other remaining project criteria.

10. Conclusion & Future Work

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. For example, transfer of arbitrary binary data can be achieved through transcoding to base64. Such infrastructure could be used to transfer arbitrarily large files, or to establish secure connections using cryptographic transport protocols such as Transport Layer Security (TLS).

11. Security Considerations

Messages sent using this system have no protection against inspection, tampering or outright forgery. The server sees all messages that are sent through the use of this service. 'Private' messaging may be easily intercepted by a 3rd party that is able to capture network traffic. Users wishing to use this system for secure communication should use/implement their own user-to-user encryption protocol.

12. IANA Considerations

None

12.1. 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.

13. Acknowledgments

This document was prepared using 2-Word-v2.0.template.dot.

Authors' Addresses