# Internet Relay Chat Protocol

Abstract

The IRC protocol is a text-based protocol, with the simplest client being any socket program capable of connecting to the server.

Table of Contents

## 1. INTRODUCTION

This document describes the IRC (Internet Relay Chat) protocol for a client-server chat application. In this application, the clients communicate with each other indirectly by sending a message to a server, and get the responses from server.

This is a basic IRC model that involves a single process (the server) forming a central point for clients to connect to, performing the required message delivery/multiplexing and other functions.

### 1.1 Server

Server provides a point to which multiple clients may connect to talk with each other. There is only one server in the network.

### 1.2 Clients

A client is a program providing a text-based interface that is used to communicate interactively by using my IRC protocol. Each client is distinguished from other clients by a unique nickname.

My IRC network contains one server and all of clients connecting to the server. There is no directly connection between clients.

```
               _______ ( Server ) _______
              /           |              \
             /            |               \
            /             |                \
           /              |                 \
          /               |                  \
     Client 1          Client 2   . . .  Client n
```

Fig. 1. Simple IRC network

### 1.3 Rooms

A room is a named group of one or more clients which will all receive messages addressed to that room. The room is created implicitly when the first client joins it, and the room will be wiped out when the last client leaves it. While room exists, any client can reference the room using the name of the room.

To send message to particular room, the client need to specify the room name. It is a string (beginning with a '#' character) of length up to 255 characters. The name has 2 parts. The first part is the character '#'. The second part may contains up to 254 characters, each of them need to be alphanumeric.

## 2. THE IRC SPECIFICATION

### 2.1 Overview

The protocol as described here is used for client to server connection.

### 2.2 Character codes

Each message exchanged between client and server contains a set of codes that are composed of eight (8) bits, making up an octet.

### 2.3 Messages

Each IRC message is a line of characters terminated with CR-LF (Carriage Return – Line Feed) pair.
The maximum length of each message is 510 characters.

#### 2.3.1 Client sending message format in 'pseudo' BNF

Each message consists of up to two main parts: the command, and the command parameters. These two parts are separated by one or more space character(s). Server will parse the message based on this format.
The BNF representation for parsing the message is:

```
<message>    ::= <command> <params> <crlf>
<command>    ::= <letter> { <letter> }

<SPACE>      ::= ' ' { ' ' }

<params>     ::= [<SPACE> [ ':' <text> | <prev_param> <params> ]]

<prev_param> ::= <Any non-empty sequence of octets not including SPACE
                 or NULL or CR or LF, the first of which may not be
                 ':'>
<text>       ::= <Any, possibly empty, sequence of octets not including
                 NUL or CR or LF>
<crlf>       ::= CR LF
<room>       ::= '#' <string>
<nick>       ::= <string>

<string>     ::= (<letter>|<number>){<letter>|<number>}

<letter>     ::= 'a'  . . .  'z' | 'A'  . . .  'Z'

<number>     ::= '0'  . . .  '9'
```

NOTES:

1) <SPACE> is consists only of SPACE character(s) (0x20). Specially notice that TABULATION, and all other control characters are considered NON-WHITE-SPACE.

2) The fact that CR and LF cannot appear in parameter string is just artifact of the message framing. This might change later.

3) The NULL character is not special in message framing, and basically could end up inside a parameter, but as it would cause extra complexities in normal string handling. Hence, NULL is not allowed within messages.

#### 2.3.2 Server sending message format in 'pseudo' BNF

```
<reply>      ::= (<ChatMessage>|<string>)<crlf>
<ChatMessage>::= <room><SPACE><nick><SPACE><':'><string>
<string>     ::= <any 8bit code except NULL, CR, LF>
```

For most message sent from client, server generates a reply back to the client. Client will base on this format to parse the reply. The list of different replies in section 5.

## 3. CLIENT MESSAGES DETAILS

### 3.1 Connection registration

On the following pages are descriptions of each client message recognized by the IRC server.
When the server received user's message, it need to parse the message completely, and returning appropriate errors. If the server encounters a fatal error while parsing a message, an error must be sent back to the client and the parsing terminated. A fatal error could be an incorrect command, or destination (room name, or nick name), not enough parameters or incorrect privileges.

#### **3.1.1** Nick message

The commands described here are used to register a connection with and IRC server and correctly disconnect.

Command: NICK
Parameters: <nickname>

NICK message is used to give user a nickname or change the previous one. If a NICK message arrives at the server is identical to nickname of some active client in the server, the server will not execute the NICK command. If NICK message is successful, then the user is connected to the server.

A <nickname> has a maximum length of ten (10) characters.

Replies:

ERR_NONICKNAMEGIVEN
ERR_NICKNAMEINUSE
ERR_ERRONEUSNICKNAME
ERR_EXCEEDNICKMAXLENGTH

Example:

```
NICK blackcat           ; Introduce new nick “blackcat”
```

#### **3.1.2** Quit message

Command: QUIT

A client session is ended with a quit message. The server must close the connection to a client who sends a QUIT message.

Numeric Replies:

None.

### **3.2** Operations

In general, these message in this section is used by a client to join, create, leave a room, and send messages to all of users in particular room. Besides that, room operators have some additional functions such as preventing normal user from changing the topic, and set a password for particular room.

#### **3.2.1** Join message

Command   : JOIN

Parameters: <room>

The JOIN command is used by client to join a room that has already been created or to create a rooms which are currently not created by any user.
If a JOIN is successful, the user is then sent the room's topic if it was set (using RPL_TOPIC) and the list of users who are on the room (using RPL_NAMREPLY).

Replies:

```
ERR_NEEDMOREPARAMS
ERR_ROOMISFULL
ERR_NOSUCHROOM
RPL_NAMEREPLY
```

Examples:

```
JOIN #hack                ; join room #hack
```

### 3.2.2 Leave message

Command   : LEAVE
Parameters: <room>

The LEAVE message is used by the client to ask the given room listed in the parameter to remove it from the list of active users.

Replies:

```
ERR_NEEDMOREPARAMS
ERR_NOTINROOM
ERR_NOSUCHROOM
```

Examples:

```
LEAVE #hack             ; leave room #hack
```

### 3.2.3 List message

Command: LIST
Parameters: <room>

The list message is used to list rooms and their topics. If <room> parameter is used, only the status of that room is displayed.

Replies:

```
ERR_NOSUCHROOM
RPL_LIST
```

Examples:

```
LIST                    ; List all rooms
LIST #hack              ; List room #hack
```

### 3.2.4 Users in a room

Command   : USERS
Parameters: <room>

The USERS command is used to view users in a room.

Replies:

```
      ERR_NEEDMOREPARAMS
      ERR_NOSUCHROOM
      RPL_USERS
```

Examples:

```
      USERS #hack ; list all users in room #hack
```

### 3.2.5 Send messages

```
Command   : SEND
Parameters: <room> <text to be sent>
```

The client uses SEND message to send a text to all clients in the same <room>.

Replies:

```
      ERR_NOTEXTTOSEND
      ERR_CANNOTSENDTOROOM
      ERR_NOSUCHROOM
```

Examples:

```
SEND #hack :testing     ; Send message “testing” to all clients in room
#hack
```

### 3.2.6 Whisper messages

```
Command   : WHISPER
Parameters: <user> <text to be sent>
```

The client uses WHISPER message to send a private text to a specific user of interest.

Replies:

```
      ERR_NOTEXTTOSEND
      ERR_CANNOTSENDTOUSER
      ERR_NOSUCHUSER
```

Examples:

```
WHISPER letu :testing   ; Send message “testing” to user named “letu”
```

### 3.2.7 Secure messages

```
Command   : SECURE
Parameters: <user> <public key> <text to be sent>
```

The client uses SECURE message to send an encrypted text to a specific user of interest using that user’s public key.

Replies:

ERR_NOTEXTTOSEND
ERR_CANNOTSENDTOUSER
ERR_NOSUCHUSER
ERR_CANNOTENCRYPT

Examples:

SECURE letu KEY :testing ; Encrypt and whisper message “testing” to user named “letu”

### 3.2.8 File transfer

Command : FILE
Parameters: <user> <file to be sent>

The client uses FILE button to upload a file and transfer it to a specific user of interest.

Replies:

ERR_NOTHINGTOSEND
ERR_CANNOTSENDTOUSER
ERR_NOSUCHUSER

### 3.2.9 Key pair generator

Command : KEYGEN
Parameters: none

The client uses KEYGEN to generate key pair (public & private) to use for SECURE message.

Replies:

ERR_CANNOTGENERATEKEY
RPL_KEYPAIR

### 3.2.10 Decrypt messages

Command : DECRYPT
Parameters: <private key> <encrypted text>

The client uses DECRYPT to decrypt an encrypted message using corresponding private key.

Replies:

ERR_NEEDSMOREPARAMS
ERR_CANNOTDECRYPT
ERR_NOTEXTTODECRYPT
RPL_DECRYPTEDMESSAGE

### 3.2.11 Clear dialog

Command : CLEAR
Parameters: none

The client uses CLEAR to erase all contents in the window dialog.

## 4. REPLIES

The following is a list of replies that are generated in response to the commands given above.
Each reply is a string.

ERR_NONICKNAMEGIVEN
"Error: No nickname given"
- Returned when a nickname parameter expected for a command and isn't found.

ERR_NICKNAMEINUSE
"Error: Nickname <nick> is already in use"
- Returned when a NICK message is processed that results in an attempt to change to a currently existing nickname.

ERR_ERRONEUSNICKNAME
"Error: Erroneus nickname <nick>"
-Returned after receiving a NICK message that contains characters which do not fall in the defined set.

ERR_EXCEEDNICKMAXLENGTH
"Error: exceed the maximum length of a nickname"
-Return after receiving a nickname that is longer than 10 characters.
ERR_NEEDMOREPARAMS
"Error: <command> did not enough parameters"
- Returned by the server by numerous commands to indicate to the client that it didn't supply enough parameters.

ERR_ROOMISFULL
"Error: Cannot join room <room>"
- Used to indicate the given room is full.

ERR_NOSUCHROOM
"Error: No such room <room>"
- Used to indicate the given room name is invalid.

ERR_EXCEEDROOMMAXLENGTH
"Error: exceed the maximum length of a room"
- Returned after receiving a room name that is longer than 254 characters.

ERR_NOTINROOM
"Error: You're not on that room <room>"
- Returned by the server whenever a client tries to perform a room effecting command for which the client isn't a member.

ERR_NOTEXTTOSEND
"Error: No text to send"
- Returned by the server to indicate that the massage wasn't delivered because it is NULL.

ERR_CANNOTSENDTOROOM
"Error: Cannot send to room <room>"
- Returned because the user is not active in this room.

RPL_NAMEREPLY
"<room> [<nick> [<nick> [...]]]"
- Returned after client join a room or create a new room.

RPL_LIST

```
        “Available rooms: <room names>”
        - The output of LIST command.

  RPL_KEYPAIR
        “Private Key=...
        Public Key=...”
        - The output of KEYGEN command.

  RPL_DECRYPTEDMESSAGE
        “Decrypted message=...”
        - The output of DECRYPT command.
```

**5.** Network protocol: TCP

My IRC is implemented on top of TCP, which supplies reliable data transmission, congestion control and high error handling and good failure recovery.

**6.** CLIENT-SERVER HANDSHAKE MECHANISM

After TCP handshaking finished, the client process sends the nickname entered by client to the server thread. The server thread will check whether this nick has been registered by any client or not. If the nick is not identical to any nick in the server, then the client can begin sending its messages to the server. If the nick is identical to some nick in the server, then the server will send error reply back to the client, client can try to send other nicknames until the server accepts, or send quit message to the server to terminate the connection.

```
CLIENT --------------- NICK -------------------------> SERVER
CLIENT <-------------- ERR_NICKNAMEINUSE ------------- SERVER
CLIENT --------------- NICK -------------------------> SERVER
CLIENT --------------- CONNECTED --------------------- SERVER
CLIENT --------------- QUIT -------------------------> SERVER
CLIENT <-------------- TERMINATED -------------------- SERVER
```