Vidhya Lakshmi Venkatarama
CS594: Internetworking protocols

RFC: Internet Relay Chat Protocol

Contents

Introduction

This document describes the IRC protocol that is intended to be developed in partial fulfilment for the term project of the course CS594: Internetworking Protocols. It shall be developed using python and verified on systems using TCP/IP network protocol. The main setup of our IRC shall include a server, number of clients and few channels/chat rooms.

Server

A server is the point at which various clients connect to talk to each other. The length of the name of the server is limited to 20 characters.

Client

A client is that part of the application that runs on a host, is connected to a server and requires the server's help to contact other clients on the network. Each client in the network will have a unique nick name, starting with a letter and having a maximum length of 10 characters.

Channel/Room

A channel/room is group of one or more client(s) who will receive a message sent by any client in that group, on that channel. Each channel will have a unique name, starting with the character '#' and the length of name not exceeding 50 characters. When a client tries to join a room by referencing its name, if the room doesn't exist, it will be created, else the client joins the room. A channel will be destroyed (i.e. be out of existence) when the last client in that channel leaves it. Further, the number of clients joining a room will be limited to 50 clients.

IRC Specification

Messages

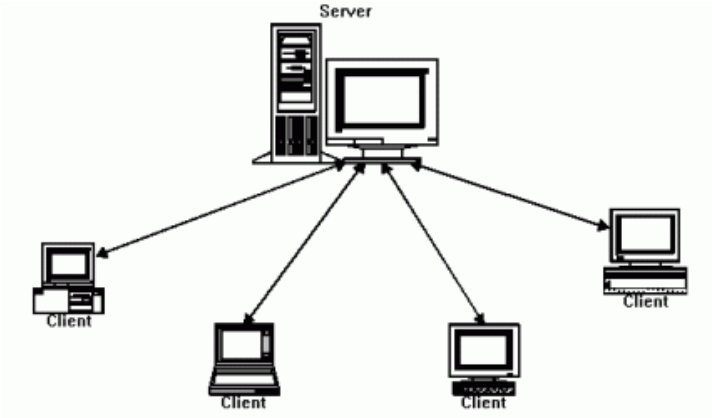
The client and the server send messages to each other and their validity will be checked. The client does not wait for a reply. The messages passed will be lines of characters, terminated by a carriage return-line feed pair. The length of the messages will be limited to 500 characters including CR-LF.

Message replies

The reply to a message will be a three-digit numeric reply. However, a numeric reply from a client to the server will be dropped.

IRC Concepts

This implementation of an Internet relay Chat includes the basic functionalities such as creation of a channel, joining a channel, leaving the channel and listing the available channels. A client is able to connect to the server, join a room, list the members of the room, send distinct messages to distinct rooms and disconnect from the room/server.



Private Conversation

A client is able to communicate with other clients individually through private chats. In such a scenario, a message sent by one client is forwarded to the other by the server.

Conferencing

Efficient conferencing among clients is the basis for this implementation of the IRC. Any message from a client to a group is first sent to the server which forwards this message to all other clients in the group. If a group has only one client, then the message sent by that client is sent to the server and not routed anywhere after that. On occasion of only two clients in a group, the flow of message is similar to that in case of private conversation. When there are many clients in a room (the usual case for chat rooms), the message sent by a client reaches the server and the server individually forwards it to each distinct client in the channel.

Message Details

This section contains the description of the messages that are recognized by the IRC clients and the servers. The server connected to the client parses the messages from a client and return appropriate errors if any. In case of fatal errors, the parsing is terminated after returning error to the client. Each parameter passed with a message is validated and the client is notified appropriately.

Connection Registration

These are the commands that need to be issued in order to register/disconnect a connection between a server and a client. For maintaining secured connections, all server connections will have a password.

Connect Message

Command: CONN
Parameters: <nickname><server><port>

This message is used to register a connection between the server and the client. This command needs to be executed together with starting the client. The client has to mention its unique nickname, the name of the server it wants to connect to and the server process' port. If the server detects that the nickname is already registered, it returns an error message.

Reply:
RPL_CONNECTED ERR_NICKCOLLISION
Example:
CONN compli server1 5678

Client Quit Message

Command: QUIT
Parameters: <Quit Message>

The client session ends with a quit message. The server closes the connection to a client that issued a quit message. In cases when a quit message is not mentioned, the server requests more parameters. If client connection is closed without client issuing a quit message, the server fills in the quit message with the nature of event leading to the situation. No replies are provided for this message.

Example: QUIT going to sleep

Server Quit Message

Command: SQUIT
Parameters: <comment>

This message is issued by a server when it quits/ disconnects from all the clients it is connected to. This message is propagated to all clients who in turn realise that the server has quit and hence they can also quit. No replies are provided for this message.

Message: RPL_SQUIT

Example: SQUIT Server Timeout

Server Disconnect Client Message

Command: DISCONN
Parameters: <nickname>

This message is issued by a server when it quits/ disconnects from all the clients it is connected to. This message is propagated to all clients who in turn realise that the server has quit and hence they can also quit. No replies are provided for this message.

Message: RPL_DISCONNECTED

Example: DISCONN compli

Channel Operations

This set of operations handle the IRC channels.

Join Command

Command: JOIN
Parameters: <channel> [<key>]

A client uses the JOIN command to start listening to a particular channel. On receipt of a JOIN command from a client, the server would check the existence of the channel, the validity of the key and whether the client is allowed to join the channel. If the channel doesn't exist, then a new channel is created. On connecting to a channel, the clients already on the channel receives a notification about this client joining it.

Replies:
ERR_NEEDMOREPARAMS
ERR_ALREADYONCHANNEL
ERR_BADCHANNELKEY
ERR_CHANNELISSUE
ERR_MAXUSERS

Examples:
JOIN #channel1
JOIN #channel1 key1

Leave Command

Command: LEAVE
Parameters: <channel>

On execution of the LEAVE command, the client sending the message is removed from the channel mentioned in the parameter. Further, the details of the client (nickname, IP address and port) are also removed from the list of active users for all clients in that channel and these clients are notified of the client leaving the channel.

Replies:
ERR_NEEDMOREPARAMS ERR_NOSUCHCHANNEL
ERR_NOTONCHANNEL

Example:
LEAVE #channel1

Topic Message

Command: TOPIC
Parameters: <channel> [<topic>]

The TOPIC message is sent by a client to change the topic of the channel mentioned If there is a topic mentioned in the message, the 'topic' of the group is changed to <topic>, else it is changed to None.

Replies:
ERR_NEEDMOREPARAMS ERR_NOTONCHANNEL
RPL_TOPIC ERR_NOSUCHCHANNEL

Examples:
TOPIC #channel1
TOPIC #channel1 topic1

List Command

Command: LIST
Parameters: [<channel>]

A client uses a LIST message to list all the channels available (existing) on the server along with their topics. If the parameter <channel> is used, only the topic of that channel is displayed (i.e. 'topic' if channel exists or error message if channel doesn't exist).

Replies:
ERR_NOCHANNELS ERR_NOSUCHCHANNEL
RPL_TOPIC RPL_LISTSTART
RPL_LIST RPL_LISTEND

Examples:
LIST
LIST #channel1

List Members

Command: LISTMEM
Parameters: <channel>

By using the LISTMEM command, the user of a channel can list the nickname of all the members in that particular channel. To do this, the client should also be a member of the channel.

Replies:

ERR_NEEDMOREPARAMS	ERR_NOSUCHCHANNEL
ERR_NOTONCHANNEL	RPL_NAMES
RPL_LISTSTART	RPL_LISTEND

Example:
LISTMEM #channel1

Sending Messages

The basic functionality of this chat protocol is to enable communication amongst the clients. The two types of communications, as we briefly touched upon earlier are a private chat and a group chat (using channels/rooms).

Private Messages

Command: PRIVMSG
Parameters: <receiver nickname> <text to be sent>

As the name describes, this command is used by clients to send private message to one other client in the application, specified by the nickname parameter.

Replies:

ERR_NEEDMOREPARAMS	ERR_NOSUCHNICK
ERR_NOTEXTTOSEND	ERR_SELFMSG

Examples:
PRIVMSG compli : This is a test message!

Group Message

Command: GRPMSG
Parameters: <channel><text>

The group message is similar to private message except it is sent to all the members of a group/channel, the sender is connected to.

Replies:

ERR_NEEDMOREPARAMS	ERR_NOTONCHANNEL
ERR_NOSUCHCHANNEL	ERR_NOTEXTTOSEND

Example:
GRPMSG #channel1: This is a test message.

REPLIES

The following is a list of numeric replies which are generated in response to the commands given in the messages section.

Error Replies

ERR_NEEDMOREPARAMS
“Need more parameters”
Used to indicate that the client didn’t supply enough parameters as needed by the command.

ERR_NICKCOLLISION
“Nickname is already in use”
Used to indicate that the nickname specified already exists and cannot be registered again.

ERR_NOSUCHNICK
“<nickname>: No such nick”
Used to indicate the nickname parameter supplied to a command is currently unused.

ERR_NICKISSUE
“First character of a nick name should be a letter and the length of nickname should be less than 10”

ERR_NOSUCHCHANNEL
“There is no such channel”
Used to indicate the given channel name is invalid.

ERR_CHANNELISSUE
“Channel name should begin with #”
Used to indicate the given channel name is invalid.

ERR_BADCHANNELKEY
“Invalid Key: Cannot join channel <channel>”
Returned when the <key> entered along with channel is not the correct key.

ERR_NOTONCHANNEL
“You are not on channel <channel>”
Returned when a client tries to perform a command on a channel on which the client is not a user.

ERR_ALREADYONCHANNEL
“'You are already on channel <channel>”

ERR_BANNEDFROMCHANNEL
“<channel>: Cannot join channel”
Returned when the client is not allowed to connect to a channel

ERR_NOTEXTTOSEND
“No text to send”
Returned when there is no message associated with the commands that need it.

ERR_NOSUCHCOMMAND
“Please enter a valid command”

ERR_MAXUSERS
“Maximum number of users in channel <channel>, try again later”

ERR_SELFMSG
“Hey! You can’t send yourself a message”

Command Responses

RPL_TOPIC
“The topic of the channel <channel> has been set to <topic>”
Displays the topic of the channel

RPL_NOTOPIC
“No topic yet on channel <channel>”

RPL_NOCHANNELS
“No channels yet”

RPL_LISTSTART
“List Start”
Marks the beginning of list of channels

RPL_LIST
“<channel>”
Displays the available channels

RPL_LISTEND
“List End”
Marks the end of channel list

RPL_NAMES
“<nickname>”
Displays all the active users in the channel

RPL_CONNECTED
“You are connected to chatServer on <port>”

RPL_DISCONNECTED
“Server is disconnecting you’

RPL_SQUIT
“Server is quitting: SQUIT <msg>”

Other Details

Network Protocol: TCP

This implementation of IRC uses TCP in order to provide a reliable data transfer between the clients, both in case of private messages and conferencing. Moreover, we talk about connections between clients and a server. Also there is a need to detect unresponsiveness of a connection between the server and client. This requires a connection oriented protocol.

Terminating Connections

When a client connection closes, a QUIT message as described is sent to the server, on behalf of the client. When the server exits an SQUIT message is propagated to each of all the clients connected to the server.