Internet Relay Chat Protocol

CS594
Yu Yang
yyang@pdx.edu

Abstract
This IRC protocol is designed for GUI-based client-server application. It supports some basic functionality such as creating a room, joining a room, leaving a room and listing rooms available, changing nickname and simple file transfer.

1. INTRODUCTION

This document describes the implementation of IRC application. IRC itself is a teleconferencing system using TCP/IP protocol, which (through the use of the client-server model) is well-suited to running on many machines in a distributed fashion. A typical setup 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 Servers

The server forms the backbone of IRC, providing a point to which clients may connect to to talk to each other.

1.2 Clients

Each client is distinguished from other clients by a unique nickname having a maximum length of nine (9) characters. In addition to the nickname, all servers must have the following information about all clients: the real name (or maybe IP address) of the host that the client is running on, the username of the client on that host, and the server to which the client is connected.

1.2.1 Operators

To allow a reasonable amount of order to be kept within the IRC network, a special class of clients (operators) is allowed to perform general maintenance functions on the network.
Operators should be able to perform basic network tasks such as disconnecting and reconnecting to servers as needed to prevent long-term use of bad network routing.

A more controversial power of operators is the ability to remove a user from the connected network by 'force', i.e. operators are able to close the connection between any client and server. The justification for this is delicate since its abuse is both destructive and annoying.

1.3 Channels

A channel is a named group of one or more clients which will all receive messages addressed to that channel. The channel is created implicitly when the first client joins it,

and the channel ceases to exist when the last client leaves it. While channel exists, any client can reference the channel using the name of the channel.

Channel names are strings of length up to 50 characters, which may not contain any spaces (‘ ’) or other symbols such as a comma.

To create a new channel or become part of an existing channel, a user is required to JOIN the channel. If the channel doesn't exist prior to joining, the channel is created.

2. The IRC Specification

2.1 Messages

Servers and clients send each other messages which may or may not generate a reply. If the message contains a valid command, the client should expect a reply but it is not advised to wait forever for the reply.

Each IRC message may consist of two main parts: the command and the command parameters, which are separated by one ASCII space character. IRC messages are always lines of characters terminated with a CR-LF pair, and these messages shall not exceed 512 characters in length, counting all characters including the trailing CR-LF. Thus, there are 510 characters maximum allowed for the command and its parameters.

3. IRC Concepts

3.1 One-to-one communication

For one-to-one communication (aka “client – server – client communication”), client will send a message to its server, and server will parse that message to find the point that client talks to, and then forward the message to the target client.

3.2 One-to-many communication

This is implemented by groups (channels). The client sends a message to its server first, and server figures out all available clients in that channel, and then forwards the message through each client- server connection.

4. Message details

On the following pages are the descriptions of each message recognized by the IRC server and client. When the reply “Unable to connect to the server specified” is received, it means that the server could not be found or you may configure the wrong port to connect to the server. The server to which a client is connected is required to parse the complete message, returning any 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 may be considered to be incorrect command, a destination which is unknown to the server,…,etc. If a full set of parameters is presented, then each must be checked for validity and appropriate responses sent back to the client.

4.1 Connection Registration

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

4.1.1 Nick message

Nick message is used to change user's nickname and it consists of 3 parts: Nick, old nickname, new nickname. These three parts are separated by an ASCII space character. If a NICK message arrives at a server which already knows about an identical nickname for another client, a nickname collision occurs. As a result of a nickname collision, the server sends the NAME_COLLISION message back to the client and asks the client to change another nickname.

4.1.2 User message

The USER message is used at the beginning of connection to specify the username, hostname (or IP address) and the channel you want to join, and send request to the server directly. When the client send User message to the server, the server will check all its user list for the specific channel and will reply "ERROR" if name collision occurs.

4.1.3 UserLeft

A client session is ended with a UserLeft message. When the server receives a UserLeft message, it must close the connection to the client which sends the message, remove that client from its users' list, broadcast a message to tell others this client is down (Do not try to contact with it) and update each client's buddy list.

4.2 Channel operations

This group of messages is concerned with manipulating channels, their properties and their contents.

4.2.1 SEND_FILE & SEND_COMPLETE

This message is used to transfer a file between two clients. It consists of 4 parts: SEND_FILE, the user you want to send to, file name, and file size. When the server receives this message, it will parse the message and get the user you want to send, file name and prepare for receiving file data. After the client has sent the whole file, it will then send SEND_COMPLETE message to tell the server there will be no more data transferred. When the server receives SEND_COMPLETE message, it will figure out the target client and forward the whole file data to that client through SEND_FILE and SEND_COMPLETE message. Moreover, in order to make transmission more efficient, the large file will be transmitted by chunks (set to 1500 bytes each time).

4.2.2 List message

The list message is used to list all channels and all users. The available channels and active users list will be sent from server by a string starting with List at the beginning

and display them in two different combobox component. Note that for each client, it can only see those clients that are in the same channel with it.

4.3 Sending messages

The common chat messages consist of header, the user you send to, status, expression, message body. When a message is sent to the server, it will be identified by parsing its header (chatMsg). The status will be used when a private message is delivered. In this case, two clients in the same channel can talk with each other privately. (Even in the same channel, other clients cannot see their messages.)

5. Operations in server side

So far, the server side keeps a list of all active users, logs all messages and is able to send system message with systemMsg at the beginning of the string to all its clients. When the server tries to stop service, all its clients should be notified and received a serviceClose message.

6. Client and server authentication

Clients and server are both subject to the same level of authentication. For both, an IP number to hostname lookup (and reverse check on this) is performed for all connections made to the server. Both connections are then subject to a password check (if there is a password set for that connection). And the connections will be created only if the password check is passed.

7. Implementations

7.1. Network protocol: TCP

IRC has been implemented on top of TCP since TCP supplies a reliable network protocol which is well suited to this application.

7.2 Message delivery

To provide useful network IO for clients and servers, each connection is given its own private 'input stream' and ‘output stream’ in which the messages or data are sent to output stream and received from input stream. And the server will monitor the input stream to see if there is any message or data coming in.

7.3 Connection 'Liveness'

To detect when a connection has died or become unresponsive, the server must ping each of its connections that it doesn't get a response from in a given amount of time.
If a connection doesn't respond in time, its connection is closed.