Network Working Group
Request for Comments: 1459

J. Oikarinen
D. Reed

May 1993

# Internet Relay Chat Protocol

## Status of This Memo

This memo defines an Experimental Protocol for the Internet community. Discussion and suggestions for improvement are requested. Please refer to the current edition of the "IAB Official Protocol Standards" for the standardization state and status of this protocol. Distribution of this memo is unlimited.

## Abstract

The IRC protocol was developed over the last 4 years since it was first implemented as a means for users on a BBS to chat amongst themselves. Now it supports a world-wide network of servers and clients, and is stringing to cope with growth. Over the past 2 years, the average number of users connected to the main IRC network has grown by a factor of 10.

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

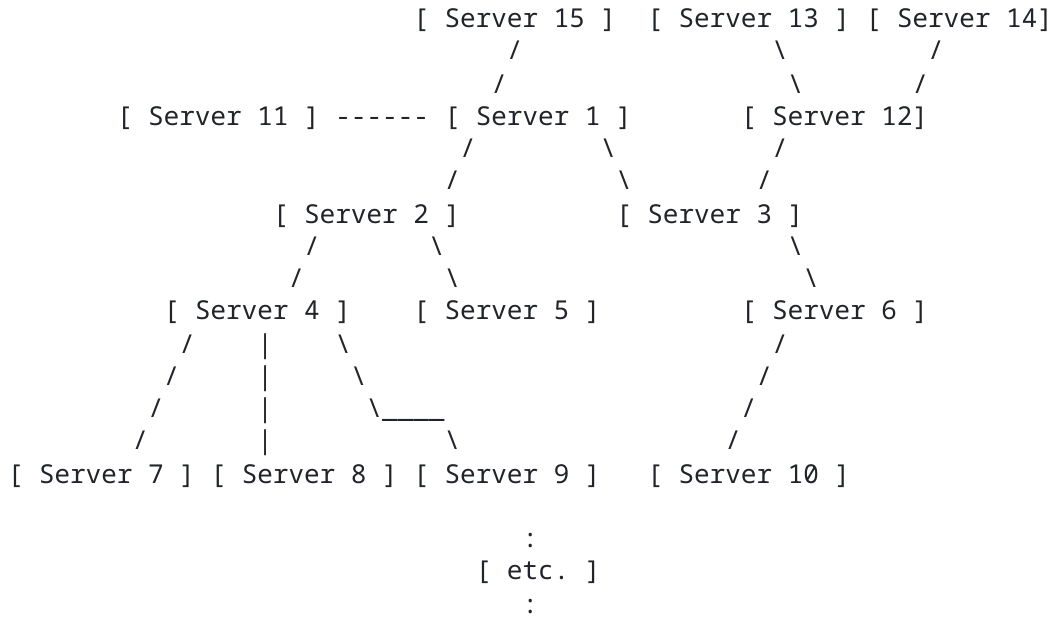
The IRC (Internet Relay Chat) protocol has been designed over a number of years for use with text based conferencing. This document describes the current IRC protocol.

The IRC protocol has been developed on systems using the TCP/IP network protocol, although there is no requirement that this remain the only sphere in which it operates.

IRC itself is a teleconferencing system, 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 (or other servers) 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, and a point for other servers to connect to, forming an IRC network. The only network configuration allowed for IRC servers is that of a spanning tree [see Fig. 1] where each server acts as a central node for the rest of the net it sees.




[ Fig. 1. Format of IRC server network ]

### 1.2 Clients

A client is anything connecting to a server that is not another server. Each client is distinguished from other clients by a unique nickname having a maximum length of nine (9) characters. See the protocol grammar rules for what may and may not be used in a nickname. In addition to the nickname, all servers must have the following information about all clients: the real name 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. Although the powers granted to an operator can be considered as 'dangerous', they are nonetheless required. Operators should be able to perform basic network tasks such as disconnecting and reconnecting servers as needed to prevent long-term use of bad network routing. In recognition of this need, the protocol discussed herein provides for operators only to be able to perform such functions. See sections 4.1.7 (SQUIT) and 4.3.5 (CONNECT).

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. For further details on this type of action, see section 4.6.1 (KILL).

### 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.

Channels names are strings (beginning with a '&' or '#' character) of length up to 200 characters. Apart from the the requirement that the first character being either '&' or '#'; the only restriction on a channel name is that it may not contain any spaces (' '), a control G (^G or ASCII 7), or a comma (',' which is used as a list item separator by the protocol).

There are two types of channels allowed by this protocol. One is a distributed channel which is known to all the servers that are

connected to the network. These channels are marked by the first character being a only clients on the server where it exists may join it. These are distinguished by a leading '&' character. On top of these two types, there are the various channel modes available to alter the characteristics of individual channels. See section 4.2.3 (MODE command) for more details on this.

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 and the creating user becomes a channel operator. If the channel already exists, whether or not your request to JOIN that channel is honoured depends on the current modes of the channel. For example, if the channel is invite-only, (+i), then you may only join if invited. As part of the protocol, a user may be a part of several channels at once, but a limit of ten (10) channels is recommended as being ample for both experienced and novice users. See section 8.13 for more information on this.

If the IRC network becomes disjoint because of a split between two servers, the channel on each side is only composed of those clients which are connected to servers on the respective sides of the split, possibly ceasing to exist on one side of the split. When the split is healed, the connecting servers announce to each other who they think is in each channel and the mode of that channel. If the channel exists on both sides, the JOINs and MODEs are interpreted in an inclusive manner so that both sides of the new connection will agree about which clients are in the channel and what modes the channel has.

### 1.3.1 Channel Operators

The channel operator (also referred to as a "chop" or "chanop") on a given channel is considered to 'own' that channel. In recognition of this status, channel operators are endowed with certain powers which enable them to keep control and some sort of sanity in their channel. As an owner of a channel, a channel operator is not required to have reasons for their actions, although if their actions are generally antisocial or otherwise abusive, it might be reasonable to ask an IRC operator to intervene, or for the usersjust leave and go elsewhere and form their own channel.

The commands which may only be used by channel operators are:

```
KICK    - Eject a client from the channel
MODE    - Change the channel's mode
INVITE  - Invite a client to an invite-only channel (mode +i)
TOPIC   - Change the channel topic in a mode +t channel
```

A channel operator is identified by the '@' symbol next to their nickname whenever it is associated with a channel (ie replies to the NAMES, WHO and WHOIS commands).

## 2. The IRC Specification

### 2.1 Overview

The protocol as described herein is for use both with server to server and client to server connections. There are, however, more restrictions on client connections (which are considered to be untrustworthy) than on server connections.

### 2.2 Character codes

No specific character set is specified. The protocol is based on a a set of codes which are composed of eight (8) bits, making up an octet. Each message may be composed of any number of these octets; however, some octet values are used for control codes which act as message delimiters.

Regardless of being an 8-bit protocol, the delimiters and keywords are such that protocol is mostly usable from USASCII terminal and a telnet connection.

Because of IRC's scandanavian origin, the characters {}| are considered to be the lower case equivalents of the characters []\, respectively. This is a critical issue when determining the equivalence of two nicknames.

### 2.3 Messages

Servers and clients send eachother messages which may or may not generate a reply. If the message contains a valid command, as described in later sections, the client should expect a reply as specified but it is not advised to wait forever for the reply; client to server and server to server communication is essentially asynchronous in nature.

Each IRC message may consist of up to three main parts: the prefix (optional), the command, and the command parameters (of which there may be up to 15). The prefix, command, and all parameters are separated by one (or more) ASCII space character(s) (0x20).

The presence of a prefix is indicated with a single leading ASCII colon character (':', 0x3b), which must be the first character of the message itself. There must be no gap (whitespace) between the colon and the prefix. The prefix is used by servers to indicate the true

origin of the message. If the prefix is missing from the message, it is assumed to have originated from the connection from which it was received. Clients should not use prefix when sending a message from themselves; if they use a prefix, the only valid prefix is the registered nickname associated with the client. If the source identified by the prefix cannot be found from the server's internal database, or if the source is registered from a different link than from which the message arrived, the server must ignore the message silently.

The command must either be a valid IRC command or a three (3) digit number represented in ASCII text.

IRC messages are always lines of characters terminated with a CR-LF (Carriage Return - Line Feed) 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. There is no provision for continuation message lines. See section 7 for more details about current implementations.

### 2.3.1 Message format in 'pseudo' BNF

The protocol messages must be extracted from the contiguous stream of octets. The current solution is to designate two characters, CR and LF, as message separators. Empty messages are silently ignored, which permits use of the sequence CR-LF between messages without extra problems.

The extracted message is parsed into the components <prefix>, <command> and list of parameters matched either by <middle> or <trailing> components.

The BNF representation for this is:

```
<message>  ::= [':' <prefix> <SPACE> ] <command> <params> <crlf>
<prefix>   ::= <servername> | <nick> [ '!' <user> ] [ '@' <host> ]
<command>  ::= <letter> { <letter> } | <number> <number> <number>
<SPACE>    ::= ' ' { ' ' }
<params>   ::= <SPACE> [ ':' <trailing> | <middle> <params> ]

<middle>   ::= <Any *non-empty* sequence of octets not including SPACE
               or NUL or CR or LF, the first of which may not be ':'>
<trailing> ::= <Any, possibly *empty*, sequence of octets not including
                 NUL or CR or LF>

<crlf>     ::= CR LF
```

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) After extracting the parameter list, all parameters are equal, whether matched by <middle> or <trailing>. <Trailing> is just a syntactic trick to allow SPACE within parameter.

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

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

5) The last parameter may be an empty string.

6) Use of the extended prefix (['!' <user> ] ['@' <host> ]) must not be used in server to server communications and is only intended for server to client messages in order to provide clients with more useful information about who a message is from without the need for additional queries.

Most protocol messages specify additional semantics and syntax for the extracted parameter strings dictated by their position in the list. For example, many server commands will assume that the first parameter after the command is the list of targets, which can be described with:

```
<target>     ::= <to> [ "," <target> ]
<to>         ::= <channel> | <user> '@' <servername> | <nick> | <mask>
<channel>    ::= ('#' | '&') <chstring>
<servername> ::= <host>
<host>       ::= see RFC 952 [DNS:4] for details on allowed hostnames
<nick>       ::= <letter> { <letter> | <number> | <special> }
<mask>       ::= ('#' | '$') <chstring>
<chstring>   ::= <any 8bit code except SPACE, BELL, NUL, CR, LF and
                  comma (',')>
```

Other parameter syntaxes are:

```
<user>       ::= <nonwhite> { <nonwhite> }
<letter>     ::= 'a' ... 'z' | 'A' ... 'Z'
<number>     ::= '0' ... '9'
<special>    ::= '-' | '[' | ']' | '\' | '`' | '^' | '{' | '}'
```

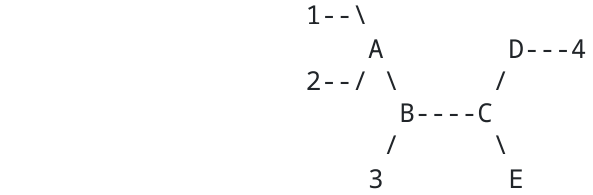
```
<nonwhite>   ::= <any 8bit code except SPACE (0x20), NUL (0x0), CR
                 (0xd), and LF (0xa)>
```

### 2.4 Numeric replies

Most of the messages sent to the server generate a reply of some sort. The most common reply is the numeric reply, used for both errors and normal replies. The numeric reply must be sent as one message consisting of the sender prefix, the three digit numeric, and the target of the reply. A numeric reply is not allowed to originate from a client; any such messages received by a server are silently dropped. In all other respects, a numeric reply is just like a normal message, except that the keyword is made up of 3 numeric digits rather than a string of letters. A list of different replies is supplied in section 6.

## 3. IRC Concepts.

This section is devoted to describing the actual concepts behind the organization of the IRC protocol and how the current implementations deliver different classes of messages.




Servers: A, B, C, D, E Clients: 1, 2, 3, 4

[ Fig. 2. Sample small IRC network ]

### 3.1 One-to-one communication

Communication on a one-to-one basis is usually only performed by clients, since most server-server traffic is not a result of servers talking only to each other. To provide a secure means for clients to talk to each other, it is required that all servers be able to send a message in exactly one direction along the spanning tree in order to reach any client. The path of a message being delivered is the shortest path between any two points on the spanning tree.

The following examples all refer to Figure 2 above.

Example 1:
A message between clients 1 and 2 is only seen by server A, which sends it straight to client 2.

Example 2:
A message between clients 1 and 3 is seen by servers A & B, and client 3. No other clients or servers are allowed see the message.

Example 3:
A message between clients 2 and 4 is seen by servers A, B, C & D and client 4 only.

### 3.2 One-to-many

The main goal of IRC is to provide a forum which allows easy and efficient conferencing (one to many conversations). IRC offers several means to achieve this, each serving its own purpose.

#### 3.2.1 To a list

The least efficient style of one-to-many conversation is through clients talking to a 'list' of users. How this is done is almost self explanatory: the client gives a list of destinations to which the message is to be delivered and the server breaks it up and dispatches a separate copy of the message to each given destination. This isn't as efficient as using a group since the destination list is broken up and the dispatch sent without checking to make sure duplicates aren't sent down each path.

#### 3.2.2 To a group (channel)

In IRC the channel has a role equivalent to that of the multicast group; their existence is dynamic (coming and going as people join and leave channels) and the actual conversation carried out on a channel is only sent to servers which are supporting users on a given channel. If there are multiple users on a server in the same channel, the message text is sent only once to that server and then sent to each client on the channel. This action is then repeated for each client-server combination until the original message has fanned out and reached each member of the channel.

The following examples all refer to Figure 2.

Example 4:
Any channel with 1 client in it. Messages to the channel go to the server and then nowhere else.

Example 5:
2 clients in a channel. All messages traverse a path as if they were private messages between the two clients outside a channel.

Example 6:
Clients 1, 2 and 3 in a channel. All messages to the channel are sent to all clients and only those servers which must be traversed by the message if it were a private message to a single client. If client 1 sends a message, it goes back to client 2 and then via server B to client 3.

### 3.2.3 To a host/server mask

To provide IRC operators with some mechanism to send messages to a large body of related users, host and server mask messages are provided. These messages are sent to users whose host or server information match that of the mask. The messages are only sent to locations where users are, in a fashion similar to that of channels.

## 3.3 One-to-all

The one-to-all type of message is better described as a broadcast message, sent to all clients or servers or both. On a large network of users and servers, a single message can result in a lot of traffic being sent over the network in an effort to reach all of the desired destinations.

For some messages, there is no option but to broadcast it to all servers so that the state information held by each server is reasonably consistent between servers.

### 3.3.1 Client-to-Client

There is no class of message which, from a single message, results in a message being sent to every other client.

### 3.3.2 Client-to-Server

Most of the commands which result in a change of state information (such as channel membership, channel mode, user status, etc) must be sent to all servers by default, and this distribution may not be changed by the client.

### 3.3.3 Server-to-Server.

While most messages between servers are distributed to all 'other' servers, this is only required for any message that affects either a user, channel or server. Since these are the basic items found in

IRC, nearly all messages originating from a server are broadcast to all other connected servers.

## 4. Message details

On the following pages are descriptions of each message recognized by the IRC server and client. All commands described in this section must be implemented by any server for this protocol.

Where the reply ERR_NOSUCHSERVER is listed, it means that the <server> parameter could not be found. The server must not send any other replies after this for that command.

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 otherwise unknown to the server (server, nick or channel names fit this category), not enough parameters or incorrect privileges.

If a full set of parameters is presented, then each must be checked for validity and appropriate responses sent back to the client. In the case of messages which use parameter lists using the comma as an item separator, a reply must be sent for each item.

In the examples below, some messages appear using the full format:

:Name COMMAND parameter list

Such examples represent a message from "Name" in transit between servers, where it is essential to include the name of the original sender of the message so remote servers may send back a reply along the correct path.

### 4.1 Connection Registration

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

A "PASS" command is not required for either client or server connection to be registered, but it must precede the server message or the latter of the NICK/USER combination. It is strongly recommended that all server connections have a password in order to give some level of security to the actual connections. The recommended order for a client to register is as follows:

1. Pass message
2. Nick message
3. User message

### 4.1.1 Password message

```
   Command: PASS
Parameters: <password>
```

The PASS command is used to set a 'connection password'. The password can and must be set before any attempt to register the connection is made. Currently this requires that clients send a PASS command before sending the NICK/USER combination and servers *must* send a PASS command before any SERVER command. The password supplied must match the one contained in the C/N lines (for servers) or I lines (for clients). It is possible to send multiple PASS commands before registering but only the last one sent is used for verification and it may not be changed once registered. Numeric Replies:

ERR_NEEDMOREPARAMS ERR_ALREADYREGISTRED

Example:

```
PASS secretpasswordhere
```

### 4.1.2 Nick message

```
   Command: NICK
Parameters: <nickname> [ <hopcount> ]
```

NICK message is used to give user a nickname or change the previous one. The <hopcount> parameter is only used by servers to indicate how far away a nick is from its home server. A local connection has a hopcount of 0. If supplied by a client, it must be ignored.

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, all instances of the nickname are removed from the server's database, and a KILL command is issued to remove the nickname from all other server's database. If the NICK message causing the collision was a nickname change, then the original (old) nick must be removed as well.

If the server recieves an identical NICK from a client which is directly connected, it may issue an ERR_NICKCOLLISION to the local client, drop the NICK command, and not generate any kills.

Numeric Replies:

ERR_NONICKNAMEGIVEN ERR_ERRONEUSNICKNAME
ERR_NICKNAMEINUSE ERR_NICKCOLLISION

Example:

```
NICK Wiz                        ; Introducing new nick "Wiz".

:WiZ NICK Kilroy                ; WiZ changed his nickname to Kilroy.
```

### 4.1.3 User message

```
   Command: USER
Parameters: <username> <hostname> <servername> <realname>
```

The USER message is used at the beginning of connection to specify the username, hostname, servername and realname of s new user. It is also used in communication between servers to indicate new user arriving on IRC, since only after both USER and NICK have been received from a client does a user become registered.

Between servers USER must to be prefixed with client's NICKname. Note that hostname and servername are normally ignored by the IRC server when the USER command comes from a directly connected client (for security reasons), but they are used in server to server communication. This means that a NICK must always be sent to a remote server when a new user is being introduced to the rest of the network before the accompanying USER is sent.

It must be noted that realname parameter must be the last parameter, because it may contain space characters and must be prefixed with a colon (':') to make sure this is recognised as such.

Since it is easy for a client to lie about its username by relying solely on the USER message, the use of an "Identity Server" is recommended. If the host which a user connects from has such a server enabled the username is set to that as in the reply from the "Identity Server".

Numeric Replies:

ERR_NEEDMOREPARAMS ERR_ALREADYREGISTRED

Examples:

```
USER guest tolmoon tolsun :Ronnie Reagan
```

```
                                ; User registering themselves with a
                                username of "guest" and real name
                                "Ronnie Reagan".


:testnick USER guest tolmoon tolsun :Ronnie Reagan
                                ; message between servers with the
                                nickname for which the USER command
                                belongs to
```

### 4.1.4 Server message

```
   Command: SERVER
Parameters: <servername> <hopcount> <info>
```

The server message is used to tell a server that the other end of a new connection is a server. This message is also used to pass server data over whole net. When a new server is connected to net, information about it be broadcast to the whole network. <hopcount> is used to give all servers some internal information on how far away all servers are. With a full server list, it would be possible to construct a map of the entire server tree, but hostmasks prevent this from being done.

The SERVER message must only be accepted from either (a) a connection which is yet to be registered and is attempting to register as a server, or (b) an existing connection to another server, in which case the SERVER message is introducing a new server behind that server.

Most errors that occur with the receipt of a SERVER command result in the connection being terminated by the destination host (target SERVER). Error replies are usually sent using the "ERROR" command rather than the numeric since the ERROR command has several useful properties which make it useful here.

If a SERVER message is parsed and attempts to introduce a server which is already known to the receiving server, the connection from which that message must be closed (following the correct procedures), since a duplicate route to a server has formed and the acyclic nature of the IRC tree broken.

Numeric Replies:

```
        ERR_ALREADYREGISTRED
```

Example:

```
SERVER test.oulu.fi 1 :[tolsun.oulu.fi] Experimental server
                                ; New server test.oulu.fi introducing
                                itself and attempting to register.  The
                                name in []'s is the hostname for the
                                host running test.oulu.fi.


:tolsun.oulu.fi SERVER csd.bu.edu 5 :BU Central Server
                                ; Server tolsun.oulu.fi is our uplink
                                for csd.bu.edu which is 5 hops away.
```

### 4.1.5 Oper

```
      Command: OPER
   Parameters: <user> <password>

   OPER message is used by a normal user to obtain operator privileges.
   The combination of <user> and <password> are required to gain
   Operator privileges.

   If the client sending the OPER command supplies the correct password
   for the given user, the server then informs the rest of the network
   of the new operator by issuing a "MODE +o" for the clients nickname.

   The OPER message is client-server only.

   Numeric Replies:

           ERR_NEEDMOREPARAMS              RPL_YOUREOPER
           ERR_NOOPERHOST                  ERR_PASSWDMISMATCH

   Example:

   OPER foo bar                    ; Attempt to register as an operator
                                   using a username of "foo" and "bar" as
                                   the password.
```

### 4.1.6 Quit

```
      Command: QUIT
   Parameters: [<Quit message>]

   A client session is ended with a quit message.  The server must close
   the connection to a client which sends a QUIT message. If a "Quit
   Message" is given, this will be sent instead of the default message,
   the nickname.

   When netsplits (disconnecting of two servers) occur, the quit message
```

is composed of the names of two servers involved, separated by a space. The first name is that of the server which is still connected and the second name is that of the server that has become disconnected.

If, for some other reason, a client connection is closed without the client issuing a QUIT command (e.g. client dies and EOF occurs on socket), the server is required to fill in the quit message with some sort of message reflecting the nature of the event which caused it to happen.

Numeric Replies:

None.

Examples:

```
QUIT :Gone to have lunch        ; Preferred message format.
```

### 4.1.7 Server quit message

```
   Command: SQUIT
Parameters: <server> <comment>
```

The SQUIT message is needed to tell about quitting or dead servers. If a server wishes to break the connection to another server it must send a SQUIT message to the other server, using the the name of the other server as the server parameter, which then closes its connection to the quitting server.

This command is also available operators to help keep a network of IRC servers connected in an orderly fashion. Operators may also issue an SQUIT message for a remote server connection. In this case, the SQUIT must be parsed by each server inbetween the operator and the remote server, updating the view of the network held by each server as explained below.

The <comment> should be supplied by all operators who execute a SQUIT for a remote server (that is not connected to the server they are currently on) so that other operators are aware for the reason of this action. The <comment> is also filled in by servers which may place an error or similar message here.

Both of the servers which are on either side of the connection being closed are required to to send out a SQUIT message (to all its other server connections) for all other servers which are considered to be behind that link.

Similarly, a QUIT message must be sent to the other connected servers rest of the network on behalf of all clients behind that link. In addition to this, all channel members of a channel which lost a member due to the split must be sent a QUIT message.

If a server connection is terminated prematurely (e.g. the server on the other end of the link died), the server which detects this disconnection is required to inform the rest of the network that the connection has closed and fill in the comment field with something appropriate.

Numeric replies:

ERR_NOPRIVILEGES ERR_NOSUCHSERVER

Example:

```
SQUIT tolsun.oulu.fi :Bad Link ? ; the server link tolson.oulu.fi has
                                 been terminated because of "Bad Link".

:Trillian SQUIT cm22.eng.umd.edu :Server out of control
                                  ; message from Trillian to disconnect
                                 "cm22.eng.umd.edu" from the net
                                  because "Server out of control".
```

### 4.2 Channel operations

This group of messages is concerned with manipulating channels, their properties (channel modes), and their contents (typically clients). In implementing these, a number of race conditions are inevitable when clients at opposing ends of a network send commands which will ultimately clash. It is also required that servers keep a nickname history to ensure that wherever a <nick> parameter is given, the server check its history in case it has recently been changed.

#### 4.2.1 Join message

```
   Command: JOIN
Parameters: <channel>{,<channel>} [<key>{,<key>}]
```

The JOIN command is used by client to start listening a specific channel. Whether or not a client is allowed to join a channel is checked only by the server the client is connected to; all other servers automatically add the user to the channel when it is received from other servers. The conditions which affect this are as follows:

1. the user must be invited if the channel is invite-only;

2. the user's nick/username/hostname must not match any active bans;

3. the correct key (password) must be given if it is set.

These are discussed in more detail under the MODE command (see section 4.2.3 for more details).

Once a user has joined a channel, they receive notice about all commands their server receives which affect the channel. This includes MODE, KICK, PART, QUIT and of course PRIVMSG/NOTICE. The JOIN command needs to be broadcast to all servers so that each server knows where to find the users who are on the channel. This allows optimal delivery of PRIVMSG/NOTICE messages to the channel.

If a JOIN is successful, the user is then sent the channel's topic (using RPL_TOPIC) and the list of users who are on the channel (using RPL_NAMREPLY), which must include the user joining.

Numeric Replies:

| | |
|---|---|
| ERR_NEEDMOREPARAMS | ERR_BANNEDFROMCHAN |
| ERR_INVITEONLYCHAN | ERR_BADCHANNELKEY |
| ERR_CHANNELISFULL | ERR_BADCHANMASK |
| ERR_NOSUCHCHANNEL | ERR_TOOMANYCHANNELS |
| RPL_TOPIC | |

Examples:

```
JOIN #foobar                    ; join channel #foobar.

JOIN &foo fubar                 ; join channel &foo using key "fubar".

JOIN #foo,&bar fubar            ; join channel #foo using key "fubar"
                                and &bar using no key.

JOIN #foo,#bar fubar,foobar     ; join channel #foo using key "fubar".
                                and channel #bar using key "foobar".

JOIN #foo,#bar                  ; join channels #foo and #bar.

:WiZ JOIN #Twilight_zone        ; JOIN message from WiZ
```

### 4.2.2 Part message

```
   Command: PART
Parameters: <channel>{,<channel>}
```

The PART message causes the client sending the message to be removed from the list of active users for all given channels listed in the parameter string.

Numeric Replies:

ERR_NEEDMOREPARAMS ERR_NOSUCHCHANNEL
ERR_NOTONCHANNEL

Examples:

```
PART #twilight_zone             ; leave channel "#twilight_zone"

PART #oz-ops,&group5            ; leave both channels "&group5" and
                                "#oz-ops".
```

### 4.2.3 Mode message

Command: MODE

The MODE command is a dual-purpose command in IRC. It allows both usernames and channels to have their mode changed. The rationale for this choice is that one day nicknames will be obsolete and the equivalent property will be the channel.

When parsing MODE messages, it is recommended that the entire message be parsed first and then the changes which resulted then passed on.

#### 4.2.3.1 Channel modes

Parameters: <channel> {[+|-]|o|p|s|i|t|n|b|v} [<limit>] [<user>] [<ban mask>]

The MODE command is provided so that channel operators may change the characteristics of `their' channel. It is also required that servers be able to change channel modes so that channel operators may be created.

The various modes available for channels are as follows:

```
o - give/take channel operator privileges;
p - private channel flag;
s - secret channel flag;
i - invite-only channel flag;
t - topic settable by channel operator only flag;
n - no messages to channel from clients on the outside;
m - moderated channel;
l - set the user limit to channel;
```

```
        b - set a ban mask to keep users out;
        v - give/take the ability to speak on a moderated channel;
        k - set a channel key (password).
```

When using the 'o' and 'b' options, a restriction on a total of three per mode command has been imposed. That is, any combination of 'o' and

#### 4.2.3.2 User modes

Parameters: <nickname> {[+|-]|i|w|s|o}

The user MODEs are typically changes which affect either how the client is seen by others or what 'extra' messages the client is sent. A user MODE command may only be accepted if both the sender of the message and the nickname given as a parameter are both the same.

The available modes are as follows:

```
        i - marks a users as invisible;
        s - marks a user for receipt of server notices;
        w - user receives wallops;
        o - operator flag.
```

Additional modes may be available later on.

If a user attempts to make themselves an operator using the "+o" flag, the attempt should be ignored. There is no restriction, however, on anyone `deopping' themselves (using "-o"). Numeric Replies:

```
        ERR_NEEDMOREPARAMS              RPL_CHANNELMODEIS
        ERR_CHANOPRIVSNEEDED            ERR_NOSUCHNICK
        ERR_NOTONCHANNEL                ERR_KEYSET
        RPL_BANLIST                     RPL_ENDOFBANLIST
        ERR_UNKNOWNMODE                 ERR_NOSUCHCHANNEL

        ERR_USERSDONTMATCH              RPL_UMODEIS
        ERR_UMODEUNKNOWNFLAG
```

Examples:

```
        Use of Channel Modes:

MODE #Finnish +im               ; Makes #Finnish channel moderated and
                                'invite-only'.

MODE #Finnish +o Kilroy         ; Gives 'chanop' privileges to Kilroy on
```

```
                                channel #Finnish.

MODE #Finnish +v Wiz            ; Allow WiZ to speak on #Finnish.

MODE #Fins -s                   ; Removes 'secret' flag from channel
                                #Fins.

MODE #42 +k oulu                ; Set the channel key to "oulu".

MODE #eu-opers +l 10            ; Set the limit for the number of users
                                on channel to 10.

MODE &oulu +b                   ; list ban masks set for channel.

MODE &oulu +b *!*@*             ; prevent all users from joining.

MODE &oulu +b *!*@*.edu         ; prevent any user from a hostname
                                matching *.edu from joining.

        Use of user Modes:

:MODE WiZ -w                    ; turns reception of WALLOPS messages
                                off for WiZ.

:Angel MODE Angel +i            ; Message from Angel to make themselves
                                invisible.

MODE WiZ -o                     ; WiZ 'deopping' (removing operator
                                status).  The plain reverse of this
                                command ("MODE WiZ +o") must not be
                                allowed from users since would bypass
                                the OPER command.
```

### 4.2.4 Topic message

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

   The TOPIC message is used to change or view the topic of a channel.
   The topic for channel <channel> is returned if there is no <topic>
   given.  If the <topic> parameter is present, the topic for that
   channel will be changed, if the channel modes permit this action.

   Numeric Replies:

           ERR_NEEDMOREPARAMS              ERR_NOTONCHANNEL
           RPL_NOTOPIC                     RPL_TOPIC
           ERR_CHANOPRIVSNEEDED
```

Examples:

```
:Wiz TOPIC #test :New topic     ;User Wiz setting the topic.

TOPIC #test :another topic      ;set the topic on #test to "another
                                topic".

TOPIC #test                     ; check the topic for #test.
```

### 4.2.5 Names message

```
   Command: NAMES
Parameters: [<channel>{,<channel>}]
```

By using the NAMES command, a user can list all nicknames that are visible to them on any channel that they can see. Channel names which they can see are those which aren't private (+p) or secret (+s) or those which they are actually on. The <channel> parameter specifies which channel(s) to return information about if valid. There is no error reply for bad channel names.

If no <channel> parameter is given, a list of all channels and their occupants is returned. At the end of this list, a list of users who are visible but either not on any channel or not on a visible channel are listed as being on `channel' "*".

Numerics:

```
        RPL_NAMREPLY                    RPL_ENDOFNAMES
```

Examples:

```
NAMES #twilight_zone,#42        ; list visible users on #twilight_zone
                                and #42 if the channels are visible to
                                you.

NAMES                           ; list all visible channels and users
```

### 4.2.6 List message

```
   Command: LIST
Parameters: [<channel>{,<channel>} [<server>]]
```

The list message is used to list channels and their topics. If the <channel> parameter is used, only the status of that channel is displayed. Private channels are listed (without their topics) as channel "Prv" unless the client generating the query is actually on that channel. Likewise, secret channels are not listed

at all unless the client is a member of the channel in question.

Numeric Replies:

```
        ERR_NOSUCHSERVER                RPL_LISTSTART
        RPL_LIST                        RPL_LISTEND
```

Examples:

```
LIST                            ; List all channels.

LIST #twilight_zone,#42         ; List channels #twilight_zone and #42
```

### 4.2.7 Invite message

```
   Command: INVITE
Parameters: <nickname> <channel>
```

The INVITE message is used to invite users to a channel. The parameter <nickname> is the nickname of the person to be invited to the target channel <channel>. There is no requirement that the channel the target user is being invited to must exist or be a valid channel. To invite a user to a channel which is invite only (MODE +i), the client sending the invite must be recognised as being a channel operator on the given channel.

Numeric Replies:

```
        ERR_NEEDMOREPARAMS              ERR_NOSUCHNICK
        ERR_NOTONCHANNEL                ERR_USERONCHANNEL
        ERR_CHANOPRIVSNEEDED
        RPL_INVITING                    RPL_AWAY
```

Examples:

```
:Angel INVITE Wiz #Dust         ; User Angel inviting WiZ to channel
                                #Dust

INVITE Wiz #Twilight_Zone       ; Command to invite WiZ to
                                #Twilight_zone
```

### 4.2.8 Kick command

```
   Command: KICK
Parameters: <channel> <user> [<comment>]
```

The KICK command can be used to forcibly remove a user from a channel. It 'kicks them out' of the channel (forced PART).

Only a channel operator may kick another user out of a channel. Each server that receives a KICK message checks that it is valid (ie the sender is actually a channel operator) before removing the victim from the channel.

Numeric Replies:

ERR_NEEDMOREPARAMS ERR_NOSUCHCHANNEL
ERR_BADCHANMASK ERR_CHANOPRIVSNEEDED
ERR_NOTONCHANNEL

Examples:

```
KICK &Melbourne Matthew         ; Kick Matthew from &Melbourne

KICK #Finnish John :Speaking English
                                ; Kick John from #Finnish using
                                "Speaking English" as the reason
                                (comment).

:WiZ KICK #Finnish John         ; KICK message from WiZ to remove John
                                from channel #Finnish
```

NOTE:
It is possible to extend the KICK command parameters to the following:

```
<channel>{,<channel>} <user>{,<user>} [<comment>]
```

### 4.3 Server queries and commands

The server query group of commands has been designed to return information about any server which is connected to the network. All servers connected must respond to these queries and respond correctly. Any invalid response (or lack thereof) must be considered a sign of a broken server and it must be disconnected/disabled as soon as possible until the situation is remedied.

In these queries, where a parameter appears as "<server>", it will usually mean it can be a nickname or a server or a wildcard name of some sort. For each parameter, however, only one query and set of replies is to be generated.

#### 4.3.1 Version message

Command: VERSION
Parameters: [<server>]

The VERSION message is used to query the version of the server program. An optional parameter <server> is used to query the version of the server program which a client is not directly connected to.

Numeric Replies:

ERR_NOSUCHSERVER RPL_VERSION

Examples:

```
:Wiz VERSION *.se              ; message from Wiz to check the version
                               of a server matching "*.se"

VERSION tolsun.oulu.fi         ; check the version of server
                               "tolsun.oulu.fi".
```

### 4.3.2 Stats message

```
   Command: STATS
Parameters: [<query> [<server>]]
```

The stats message is used to query statistics of certain server. If <server> parameter is omitted, only the end of stats reply is sent back. The implementation of this command is highly dependent on the server which replies, although the server must be able to supply information as described by the queries below (or similar).

A query may be given by any single letter which is only checked by the destination server (if given as the <server> parameter) and is otherwise passed on by intermediate servers, ignored and unaltered. The following queries are those found in the current IRC implementation and provide a large portion of the setup information for that server. Although these may not be supported in the same way by other versions, all servers should be able to supply a valid reply to a STATS query which is consistent with the reply formats currently used and the purpose of the query.

The currently supported queries are:

```
c - returns a list of servers which the server may connect
    to or allow connections from;
h - returns a list of servers which are either forced to be
    treated as leaves or allowed to act as hubs;
i - returns a list of hosts which the server allows a client
    to connect from;
k - returns a list of banned username/hostname combinations
    for that server;
l - returns a list of the server's connections, showing how
```

```
            long each connection has been established and the traffic
            over that connection in bytes and messages for each
            direction;
        m - returns a list of commands supported by the server and
            the usage count for each if the usage count is non zero;
        o - returns a list of hosts from which normal clients may
            become operators;
        y - show Y (Class) lines from server's configuration file;
        u - returns a string showing how long the server has been up.
```

Numeric Replies:

```
        ERR_NOSUCHSERVER
        RPL_STATSCLINE                  RPL_STATSNLINE
        RPL_STATSILINE                  RPL_STATSKLINE
        RPL_STATSQLINE                  RPL_STATSLLINE
        RPL_STATSLINKINFO               RPL_STATSUPTIME
        RPL_STATSCOMMANDS               RPL_STATSOLINE
        RPL_STATSHLINE                  RPL_ENDOFSTATS
```

Examples:

```
STATS m                         ; check the command usage for the server
                                you are connected to

:Wiz STATS c eff.org            ; request by WiZ for C/N line
                                information from server eff.org
```

### 4.3.3 Links message

```
      Command: LINKS
   Parameters: [[<remote server>] <server mask>]
```

With LINKS, a user can list all servers which are known by the server answering the query. The returned list of servers must match the mask, or if no mask is given, the full list is returned.

If <remote server> is given in addition to <server mask>, the LINKS command is forwarded to the first server found that matches that name (if any), and that server is then required to answer the query.

Numeric Replies:

```
        ERR_NOSUCHSERVER
        RPL_LINKS                       RPL_ENDOFLINKS
```

Examples:

```
LINKS *.au                      ; list all servers which have a name
                                that matches *.au;

:WiZ LINKS *.bu.edu *.edu       ; LINKS message from WiZ to the first
                                server matching *.edu for a list of
                                servers matching *.bu.edu.
```

#### 4.3.4 Time message

```
      Command: TIME
   Parameters: [<server>]
```

The time message is used to query local time from the specified server. If the server parameter is not given, the server handling the command must reply to the query.

Numeric Replies:

```
        ERR_NOSUCHSERVER                RPL_TIME
```

Examples:

```
TIME tolsun.oulu.fi             ; check the time on the server
                                "tolson.oulu.fi"

Angel TIME *.au                 ; user angel checking the time on a
                                server matching "*.au"
```

#### 4.3.5 Connect message

```
      Command: CONNECT
   Parameters: <target server> [<port> [<remote server>]]
```

The CONNECT command can be used to force a server to try to establish a new connection to another server immediately. CONNECT is a privileged command and is to be available only to IRC Operators. If a remote server is given then the CONNECT attempt is made by that server to <target server> and <port>.

Numeric Replies:

```
        ERR_NOSUCHSERVER                ERR_NOPRIVILEGES
        ERR_NEEDMOREPARAMS
```

Examples:

```
CONNECT tolsun.oulu.fi          ; Attempt to connect a server to
                                tolsun.oulu.fi
```

```
:WiZ CONNECT eff.org 6667 csd.bu.edu
                                ; CONNECT attempt by WiZ to get servers
                                eff.org and csd.bu.edu connected on port
                                6667.
```

### 4.3.6 Trace message

```
      Command: TRACE
   Parameters: [<server>]
```

TRACE command is used to find the route to specific server. Each server that processes this message must tell the sender about it by sending a reply indicating it is a pass-through link, forming a chain of replies similar to that gained from using "traceroute". After sending this reply back, it must then send the TRACE message to the next server until given server is reached. If the <server> parameter is omitted, it is recommended that TRACE command send a message to the sender telling which servers the current server has direct connection to.

If the destination given by "<server>" is an actual server, then the destination server is required to report all servers and users which are connected to it, although only operators are permitted to see users present. If the destination given by <server> is a nickname, they only a reply for that nickname is given.

Numeric Replies:

```
        ERR_NOSUCHSERVER
```

If the TRACE message is destined for another server, all intermediate servers must return a RPL_TRACELINK reply to indicate that the TRACE passed through it and where its going next.

```
        RPL_TRACELINK
```

A TRACE reply may be composed of any number of the following numeric replies.

```
        RPL_TRACECONNECTING             RPL_TRACEHANDSHAKE
        RPL_TRACEUNKNOWN                RPL_TRACEOPERATOR
        RPL_TRACEUSER                   RPL_TRACESERVER
        RPL_TRACESERVICE                RPL_TRACENEWTYPE
        RPL_TRACECLASS
```

Examples:

```
TRACE *.oulu.fi                 ; TRACE to a server matching *.oulu.fi
```

```
:WiZ TRACE AngelDust            ; TRACE issued by WiZ to nick AngelDust
```

### 4.3.7 Admin command

```
   Command: ADMIN
Parameters: [<server>]
```

The admin message is used to find the name of the administrator of the given server, or current server if <server> parameter is omitted. Each server must have the ability to forward ADMIN messages to other servers.

Numeric Replies:

```
        ERR_NOSUCHSERVER
        RPL_ADMINME                     RPL_ADMINLOC1
        RPL_ADMINLOC2                   RPL_ADMINEMAIL
```

Examples:

```
ADMIN tolsun.oulu.fi            ; request an ADMIN reply from
                                tolsun.oulu.fi

:WiZ ADMIN *.edu                ; ADMIN request from WiZ for first
                                server found to match *.edu.
```

### 4.3.8 Info command

```
   Command: INFO
Parameters: [<server>]
```

The INFO command is required to return information which describes the server: its version, when it was compiled, the patchlevel, when it was started, and any other miscellaneous information which may be considered to be relevant.

Numeric Replies:

```
        ERR_NOSUCHSERVER
        RPL_INFO                        RPL_ENDOFINFO
```

Examples:

```
INFO csd.bu.edu                 ; request an INFO reply from
csd.bu.edu

:Avalon INFO *.fi               ; INFO request from Avalon for first
                                server found to match *.fi.
```

```
   INFO Angel                      ; request info from the server that
                                   Angel is connected to.
```

### 4.4 Sending messages

The main purpose of the IRC protocol is to provide a base for clients to communicate with each other. PRIVMSG and NOTICE are the only messages available which actually perform delivery of a text message from one client to another - the rest just make it possible and try to ensure it happens in a reliable and structured manner.

#### 4.4.1 Private messages

```
      Command: PRIVMSG
   Parameters: <receiver>{,<receiver>} <text to be sent>
```

PRIVMSG is used to send private messages between users. <receiver> is the nickname of the receiver of the message. <receiver> can also be a list of names or channels separated with commas.

The <receiver> parameter may also me a host mask (#mask) or server mask ($mask). In both cases the server will only send the PRIVMSG to those who have a server or host matching the mask. The mask must have at least 1 (one) "." in it and no wildcards following the last ".". This requirement exists to prevent people sending messages to "#*" or "$*", which would broadcast to all users; from experience, this is abused more than used responsibly and properly. Wildcards are the '*' and '?' characters. This extension to the PRIVMSG command is only available to Operators.

Numeric Replies:

```
        ERR_NORECIPIENT                 ERR_NOTEXTTOSEND
        ERR_CANNOTSENDTOCHAN            ERR_NOTOPLEVEL
        ERR_WILDTOPLEVEL                ERR_TOOMANYTARGETS
        ERR_NOSUCHNICK
        RPL_AWAY
```

Examples:

```
:Angel PRIVMSG Wiz :Hello are you receiving this message ?
                                ; Message from Angel to Wiz.

PRIVMSG Angel :yes I'm receiving it !receiving it !'u>(768u+1n) .br ;
                               Message to Angel.

PRIVMSG jto@tolsun.oulu.fi :Hello !
                                ; Message to a client on server
```

```
                                tolsun.oulu.fi with username of "jto".

PRIVMSG $*.fi :Server tolsun.oulu.fi rebooting.
                                ; Message to everyone on a server which
                                has a name matching *.fi.

PRIVMSG #*.edu :NSFNet is undergoing work, expect interruptions
                                ; Message to all users who come from a
                                host which has a name matching *.edu.
```

### 4.4.2 Notice

```
      Command: NOTICE
   Parameters: <nickname> <text>
```

The NOTICE message is used similarly to PRIVMSG. The difference between NOTICE and PRIVMSG is that automatic replies must never be sent in response to a NOTICE message. This rule applies to servers too - they must not send any error reply back to the client on receipt of a notice. The object of this rule is to avoid loops between a client automatically sending something in response to something it received. This is typically used by automatons (clients with either an AI or other interactive program controlling their actions) which are always seen to be replying lest they end up in a loop with another automaton.

See PRIVMSG for more details on replies and examples.

## 4.5 User based queries

User queries are a group of commands which are primarily concerned with finding details on a particular user or group users. When using wildcards with any of these commands, if they match, they will only return information on users who are 'visible' to you. The visibility of a user is determined as a combination of the user's mode and the common set of channels you are both on.

### 4.5.1 Who query

```
      Command: WHO
   Parameters: [<name> [<o>]]
```

The WHO message is used by a client to generate a query which returns a list of information which 'matches' the <name> parameter given by the client. In the absence of the <name> parameter, all visible (users who aren't invisible (user mode +i) and who don't have a common channel with the requesting client) are listed. The same result can be achieved by using a <name> of "0" or any wildcard which

will end up matching every entry possible.

The <name> passed to WHO is matched against users' host, server, real name and nickname if the channel <name> cannot be found.

If the "o" parameter is passed only operators are returned according to the name mask supplied.

Numeric Replies:

ERR_NOSUCHSERVER
RPL_WHOREPLY RPL_ENDOFWHO

Examples:

```
WHO *.fi                    ; List all users who match against
                            "*.fi".

WHO jto* o                  ; List all users with a match against
                            "jto*" if they are an operator.
```

### 4.5.2 Whois query

```
   Command: WHOIS
Parameters: [<server>] <nickmask>[,<nickmask>[,...]]
```

This message is used to query information about particular user. The server will answer this message with several numeric messages indicating different statuses of each user which matches the nickmask (if you are entitled to see them). If no wildcard is present in the <nickmask>, any information about that nick which you are allowed to see is presented. A comma (',') separated list of nicknames may be given.

The latter version sends the query to a specific server. It is useful if you want to know how long the user in question has been idle as only local server (ie. the server the user is directly connected to) knows that information, while everything else is globally known.

Numeric Replies:

ERR_NOSUCHSERVER ERR_NONICKNAMEGIVEN
RPL_WHOISUSER RPL_WHOISCHANNELS
RPL_WHOISCHANNELS RPL_WHOISSERVER
RPL_AWAY RPL_WHOISOPERATOR
RPL_WHOISIDLE ERR_NOSUCHNICK
RPL_ENDOFWHOIS

Examples:

```
WHOIS wiz                       ; return available user information
                                about nick WiZ

WHOIS eff.org trillian          ; ask server eff.org for user
                                information about trillian
```

### 4.5.3 Whowas

```
   Command: WHOWAS
Parameters: <nickname> [<count> [<server>]]
```

Whowas asks for information about a nickname which no longer exists. This may either be due to a nickname change or the user leaving IRC. In response to this query, the server searches through its nickname history, looking for any nicks which are lexically the same (no wild card matching here). The history is searched backward, returning the most recent entry first. If there are multiple entries, up to <count> replies will be returned (or all of them if no <count> parameter is given). If a non-positive number is passed as being <count>, then a full search is done.

Numeric Replies:

```
        ERR_NONICKNAMEGIVEN             ERR_WASNOSUCHNICK
        RPL_WHOWASUSER                  RPL_WHOISSERVER
        RPL_ENDOFWHOWAS
```

Examples:

```
WHOWAS Wiz                      ; return all information in the nick
                                history about nick "WiZ";

WHOWAS Mermaid 9                ; return at most, the 9 most recent
                                entries in the nick history for
                                "Mermaid";

WHOWAS Trillian 1 *.edu         ; return the most recent history for
                                "Trillian" from the first server found
                                to match "*.edu".
```

## 4.6 Miscellaneous messages

Messages in this category do not fit into any of the above categories but are nonetheless still a part of and required by the protocol.

### 4.6.1 Kill message

```
   Command: KILL
Parameters: <nickname> <comment>
```

The KILL message is used to cause a client-server connection to be closed by the server which has the actual connection. KILL is used by servers when they encounter a duplicate entry in the list of valid nicknames and is used to remove both entries. It is also available to operators.

Clients which have automatic reconnect algorithms effectively make this command useless since the disconnection is only brief. It does however break the flow of data and can be used to stop large amounts of being abused, any user may elect to receive KILL messages generated for others to keep an 'eye' on would be trouble spots.

In an arena where nicknames are required to be globally unique at all times, KILL messages are sent whenever 'duplicates' are detected (that is an attempt to register two users with the same nickname) in the hope that both of them will disappear and only 1 reappear.

The comment given must reflect the actual reason for the KILL. For server-generated KILLs it usually is made up of details concerning the origins of the two conflicting nicknames. For users it is left up to them to provide an adequate reason to satisfy others who see it. To prevent/discourage fake KILLs from being generated to hide the identify of the KILLer, the comment also shows a 'kill-path' which is updated by each server it passes through, each prepending its name to the path.

Numeric Replies:

```
        ERR_NOPRIVILEGES                ERR_NEEDMOREPARAMS
        ERR_NOSUCHNICK                  ERR_CANTKILLSERVER
```

```
KILL David (csd.bu.edu <- tolsun.oulu.fi)
                                ; Nickname collision between csd.bu.edu
                                and tolson.oulu.fi
```

NOTE:
It is recommended that only Operators be allowed to kill other users with KILL message. In an ideal world not even operators would need to do this and it would be left to servers to deal with.

### 4.6.2 Ping message

```
   Command: PING
Parameters: <server1> [<server2>]
```

The PING message is used to test the presence of an active client at the other end of the connection. A PING message is sent at regular intervals if no other activity detected coming from a connection. If a connection fails to respond to a PING command within a set amount of time, that connection is closed.

Any client which receives a PING message must respond to <server1> (server which sent the PING message out) as quickly as possible with an appropriate PONG message to indicate it is still there and alive. Servers should not respond to PING commands but rely on PINGs from the other end of the connection to indicate the connection is alive. If the <server2> parameter is specified, the PING message gets forwarded there.

Numeric Replies:

ERR_NOORIGIN ERR_NOSUCHSERVER

Examples:

```
PING tolsun.oulu.fi             ; server sending a PING message to
                                another server to indicate it is still
                                alive.

PING WiZ                        ; PING message being sent to nick WiZ
```

### 4.6.3 Pong message

```
   Command: PONG
Parameters: <daemon> [<daemon2>]
```

PONG message is a reply to ping message. If parameter <daemon2> is given this message must be forwarded to given daemon. The <daemon> parameter is the name of the daemon who has responded to PING message and generated this message.

Numeric Replies:

ERR_NOORIGIN ERR_NOSUCHSERVER

Examples:

```
PONG csd.bu.edu tolsun.oulu.fi  ; PONG message from csd.bu.edu to
```

tolsun.oulu.fi

### 4.6.4 Error

```
   Command: ERROR
Parameters: <error message>
```

The ERROR command is for use by servers when reporting a serious or fatal error to its operators. It may also be sent from one server to another but must not be accepted from any normal unknown clients.

An ERROR message is for use for reporting errors which occur with a server-to-server link only. An ERROR message is sent to the server at the other end (which sends it to all of its connected operators) and to all operators currently connected. It is not to be passed onto any other servers by a server if it is received from a server.

When a server sends a received ERROR message to its operators, the message should be encapsulated inside a NOTICE message, indicating that the client was not responsible for the error.

Numerics:

None.

Examples:

```
ERROR :Server *.fi already exists; ERROR message to the other server
                                  which caused this error.

NOTICE WiZ :ERROR from csd.bu.edu -- Server *.fi already exists
                                  ; Same ERROR message as above but sent
                                  to user WiZ on the other server.
```

## 5. OPTIONALS

This section describes OPTIONAL messages. They are not required in a working server implementation of the protocol described herein. In the absence of the option, an error reply message must be generated or an unknown command error. If the message is destined for another server to answer then it must be passed on (elementary parsing required) The allocated numerics for this are listed with the messages below.

### 5.1 Away

```
   Command: AWAY
Parameters: [message]
```

With the AWAY message, clients can set an automatic reply string for any PRIVMSG commands directed at them (not to a channel they are on). The automatic reply is sent by the server to client sending the PRIVMSG command. The only replying server is the one to which the sending client is connected to.

The AWAY message is used either with one parameter (to set an AWAY message) or with no parameters (to remove the AWAY message).

Numeric Replies:

RPL_UNAWAY RPL_NOWAWAY

Examples:

```
AWAY :Gone to lunch.  Back in 5 ; set away message to "Gone to lunch.
                                  Back in 5".

:WiZ AWAY                       ; unmark WiZ as being away.
```

### 5.2 Rehash message

```
   Command: REHASH
Parameters: None
```

The rehash message can be used by the operator to force the server to re-read and process its configuration file.

Numeric Replies:

RPL_REHASHING ERR_NOPRIVILEGES

Examples:

```
REHASH                          ; message from client with operator
                                status to server asking it to reread its
                                configuration file.
```

### 5.3 Restart message

```
   Command: RESTART
Parameters: None
```

The restart message can only be used by an operator to force a server restart itself. This message is optional since it may be viewed as a risk to allow arbitrary people to connect to a server as an operator and execute this command, causing (at least) a disruption to service.

The RESTART command must always be fully processed by the server to which the sending client is connected and not be passed onto other connected servers.

Numeric Replies:

ERR_NOPRIVILEGES

Examples:

```
RESTART                         ; no parameters required.
```

### 5.4 Summon message

```
   Command: SUMMON
Parameters: <user> [<server>]
```

The SUMMON command can be used to give users who are on a host running an IRC server a message asking them to please join IRC. This message is only sent if the target server (a) has SUMMON enabled, (b) the user is logged in and (c) the server process can write to the user's tty (or similar).

If no <server> parameter is given it tries to summon <user> from the server the client is connected to is assumed as the target.

If summon is not enabled in a server, it must return the ERR_SUMMONDISABLED numeric and pass the summon message onwards.

Numeric Replies:

ERR_NORECIPIENT ERR_FILEERROR
ERR_NOLOGIN ERR_NOSUCHSERVER
RPL_SUMMONING

Examples:

```
SUMMON jto                      ; summon user jto on the server's host

SUMMON jto tolsun.oulu.fi       ; summon user jto on the host which a
                                server named "tolsun.oulu.fi" is
                                running.
```

### 5.5 Users

```
   Command: USERS
Parameters: [<server>]
```

The USERS command returns a list of users logged into the server in a similar format to who(1), rusers(1) and finger(1). Some people may disable this command on their server for security related reasons. If disabled, the correct numeric must be returned to indicate this.

Numeric Replies:

```
        ERR_NOSUCHSERVER                ERR_FILEERROR
        RPL_USERSSTART                  RPL_USERS
        RPL_NOUSERS                     RPL_ENDOFUSERS
        ERR_USERSDISABLED
```

Disabled Reply:

```
        ERR_USERSDISABLED
```

Examples:

```
USERS eff.org                   ; request a list of users logged in on
                                server eff.org

:John USERS tolsun.oulu.fi      ; request from John for a list of users
                                logged in on server tolsun.oulu.fi
```

### 5.6 Operwall message

```
   Command: WALLOPS
Parameters: Text to be sent to all operators currently online
```

Sends a message to all operators currently online. After implementing WALLOPS as a user command it was found that it was often and commonly abused as a means of sending a message to a lot of people (much similar to WALL). Due to this it is recommended that the current implementation of WALLOPS be used as an example by allowing and recognising only servers as the senders of WALLOPS.

Numeric Replies:

```
        ERR_NEEDMOREPARAMS
```

Examples:

```
:csd.bu.edu WALLOPS :Connect '*.uiuc.edu 6667' from Joshua; WALLOPS
                       message from csd.bu.edu announcing a
                       CONNECT message it received and acted
                       upon from Joshua.
```

### 5.7 Userhost message

```
   Command: USERHOST
Parameters: <nickname>{<space><nickname>}
```

The USERHOST command takes a list of up to 5 nicknames, each separated by a space character and returns a list of information about each nickname that it found. The returned list has each reply separated by a space.

Numeric Replies:

```
        RPL_USERHOST                    ERR_NEEDMOREPARAMS
```

Examples:

```
USERHOST Wiz Michael Marty p    ;USERHOST request for information on
                                nicks "Wiz", "Michael", "Marty" and "p"
```

### 5.8 Ison message

```
   Command: ISON
Parameters: <nickname>{<space><nickname>}
```

The ISON command was implemented to provide a quick and efficient means to get a response about whether a given nickname was currently on IRC. ISON only takes one (1) parameter: a space-separated list of nicks. For each nickname in the list that is present, the server adds that to its reply string. Thus the reply string may return empty (none of the given nicks are present), an exact copy of the parameter string (all of them present) or as any other subset of the set of nicks given in the parameter. The only limit on the number of nicks that may be checked is that the combined length must not be too large as to cause the server to chop it off so it fits in 512 characters.

ISON is only be processed by the server local to the client sending the command and thus not passed onto other servers for further processing.

Numeric Replies:

```
        RPL_ISON                ERR_NEEDMOREPARAMS
```

Examples:

```
ISON phone trillian WiZ jarlek Avalon Angel Monstah
                                ; Sample ISON request for 7 nicks.
```

## 6. REPLIES

The following is a list of numeric replies which are generated in response to the commands given above. Each numeric is given with its number, name and reply string.

### 6.1 Error Replies.

401 ERR_NOSUCHNICK
"<nickname> :No such nick/channel"

- Used to indicate the nickname parameter supplied to a command is currently unused.

402 ERR_NOSUCHSERVER
"<server name> :No such server"

- Used to indicate the server name given currently doesn't exist.

403 ERR_NOSUCHCHANNEL
"<channel name> :No such channel"

- Used to indicate the given channel name is invalid.

404 ERR_CANNOTSENDTOCHAN
"<channel name> :Cannot send to channel"

- Sent to a user who is either (a) not on a channel which is mode +n or (b) not a chanop (or mode +v) on a channel which has mode +m set and is trying to send a PRIVMSG message to that channel.

405 ERR_TOOMANYCHANNELS
"<channel name> :You have joined too many \
channels"
- Sent to a user when they have joined the maximum number of allowed channels and they try to join another channel.

406 ERR_WASNOSUCHNICK
"<nickname> :There was no such nickname"

- Returned by WHOWAS to indicate there is no history information for that nickname.

407 ERR_TOOMANYTARGETS
"<target> :Duplicate recipients. No message \

```
                  delivered"

          - Returned to a client which is attempting to send a
            PRIVMSG/NOTICE using the user@host destination format
            and for a user@host which has several occurrences.

      409 ERR_NOORIGIN
                  ":No origin specified"

          - PING or PONG message missing the originator parameter
            which is required since these commands must work
            without valid prefixes.

      411 ERR_NORECIPIENT
                  ":No recipient given (<command>)"
      412 ERR_NOTEXTTOSEND
                  ":No text to send"
      413 ERR_NOTOPLEVEL
                  "<mask> :No toplevel domain specified"
      414 ERR_WILDTOPLEVEL
                  "<mask> :Wildcard in toplevel domain"

          - 412 - 414 are returned by PRIVMSG to indicate that
            the message wasn't delivered for some reason.
            ERR_NOTOPLEVEL and ERR_WILDTOPLEVEL are errors that
            are returned when an invalid use of
            "PRIVMSG $<server>" or "PRIVMSG #<host>" is attempted.

      421 ERR_UNKNOWNCOMMAND
                  "<command> :Unknown command"

          - Returned to a registered client to indicate that the
            command sent is unknown by the server.

      422 ERR_NOMOTD
                  ":MOTD File is missing"

          - Server's MOTD file could not be opened by the server.

      423 ERR_NOADMININFO
                  "<server> :No administrative info available"

          - Returned by a server in response to an ADMIN message
            when there is an error in finding the appropriate
            information.

      424 ERR_FILEERROR
          ":File error doing <file op> on <file>"
```

- Generic error message used to report a failed file operation during the processing of a message.

431 ERR_NONICKNAMEGIVEN
":No nickname given"

- Returned when a nickname parameter expected for a command and isn't found.

432 ERR_ERRONEUSNICKNAME
"<nick> :Erroneus nickname"

- Returned after receiving a NICK message which contains characters which do not fall in the defined set. See section x.x.x for details on valid nicknames.

433 ERR_NICKNAMEINUSE
"<nick> :Nickname is already in use"

- Returned when a NICK message is processed that results in an attempt to change to a currently existing nickname.

436 ERR_NICKCOLLISION
"<nick> :Nickname collision KILL"

- Returned by a server to a client when it detects a nickname collision (registered of a NICK that already exists by another server).

441 ERR_USERNOTINCHANNEL
"<nick> <channel> :They aren't on that channel"

- Returned by the server to indicate that the target user of the command is not on the given channel.

442 ERR_NOTONCHANNEL
"<channel> :You're not on that channel"

- Returned by the server whenever a client tries to perform a channel effecting command for which the client isn't a member.

443 ERR_USERONCHANNEL
"<user> <channel> :is already on channel"

- Returned when a client tries to invite a user to a channel they are already on.

444 ERR_NOLOGIN
"<user> :User not logged in"

- Returned by the summon after a SUMMON command for a user was unable to be performed since they were not logged in.

445 ERR_SUMMONDISABLED
":SUMMON has been disabled"

- Returned as a response to the SUMMON command. Must be returned by any server which does not implement it.

446 ERR_USERSDISABLED
":USERS has been disabled"

- Returned as a response to the USERS command. Must be returned by any server which does not implement it.

451 ERR_NOTREGISTERED
":You have not registered"

- Returned by the server to indicate that the client must be registered before the server will allow it to be parsed in detail.

461 ERR_NEEDMOREPARAMS
"<command> :Not enough parameters"

- Returned by the server by numerous commands to indicate to the client that it didn't supply enough parameters.

462 ERR_ALREADYREGISTRED
":You may not reregister"

- Returned by the server to any link which tries to change part of the registered details (such as password or user details from second USER message).

463 ERR_NOPERMFORHOST
":Your host isn't among the privileged"

- Returned to a client which attempts to register with a server which does not been setup to allow connections from the host the attempted connection is tried.

```
        464     ERR_PASSWDMISMATCH
                        ":Password incorrect"

                - Returned to indicate a failed attempt at registering
                  a connection for which a password was required and
                  was either not given or incorrect.

        465     ERR_YOUREBANNEDCREEP
                        ":You are banned from this server"

                - Returned after an attempt to connect and register
                  yourself with a server which has been setup to
                  explicitly deny connections to you.

        467     ERR_KEYSET
                        "<channel> :Channel key already set"
        471     ERR_CHANNELISFULL
                        "<channel> :Cannot join channel (+l)"
        472     ERR_UNKNOWNMODE
                        "<char> :is unknown mode char to me"
        473     ERR_INVITEONLYCHAN
                        "<channel> :Cannot join channel (+i)"
        474     ERR_BANNEDFROMCHAN
                        "<channel> :Cannot join channel (+b)"
        475     ERR_BADCHANNELKEY
                        "<channel> :Cannot join channel (+k)"
        481     ERR_NOPRIVILEGES
                        ":Permission Denied- You're not an IRC operator"

                - Any command requiring operator privileges to operate
                  must return this error to indicate the attempt was
                  unsuccessful.

        482     ERR_CHANOPRIVSNEEDED
                        "<channel> :You're not channel operator"

                - Any command requiring 'chanop' privileges (such as
                  MODE messages) must return this error if the client
                  making the attempt is not a chanop on the specified
                  channel.

        483     ERR_CANTKILLSERVER
                        ":You cant kill a server!"

                - Any attempts to use the KILL command on a server
                  are to be refused and this error returned directly
                  to the client.
```

```
        491     ERR_NOOPERHOST
                        ":No O-lines for your host"

                - If a client sends an OPER message and the server has
                  not been configured to allow connections from the
                  client's host as an operator, this error must be
                  returned.

        501     ERR_UMODEUNKNOWNFLAG
                        ":Unknown MODE flag"

                - Returned by the server to indicate that a MODE
                  message was sent with a nickname parameter and that
                  the a mode flag sent was not recognized.

        502     ERR_USERSDONTMATCH
                        ":Cant change mode for other users"

                - Error sent to any user trying to view or change the
                  user mode for a user other than themselves.
```

### 6.2 Command responses.

```
        300     RPL_NONE
                        Dummy reply number. Not used.

        302     RPL_USERHOST
                        ":[<reply>{<space><reply>}]"

                - Reply format used by USERHOST to list replies to
                  the query list.  The reply string is composed as
                  follows:

                  <reply> ::= <nick>['*'] '=' <'+'|'-'><hostname>

                  The '*' indicates whether the client has registered
                  as an Operator.  The '-' or '+' characters represent
                  whether the client has set an AWAY message or not
                  respectively.

        303     RPL_ISON
                        ":[<nick> {<space><nick>}]"

                - Reply format used by ISON to list replies to the
                  query list.

        301     RPL_AWAY
                        "<nick> :<away message>"
```

```
        305     RPL_UNAWAY
                        ":You are no longer marked as being away"
        306     RPL_NOWAWAY
                        ":You have been marked as being away"

                - These replies are used with the AWAY command (if
                  allowed).  RPL_AWAY is sent to any client sending a
                  PRIVMSG to a client which is away.  RPL_AWAY is only
                  sent by the server to which the client is connected.
                  Replies RPL_UNAWAY and RPL_NOWAWAY are sent when the
                  client removes and sets an AWAY message.

        311     RPL_WHOISUSER
                        "<nick> <user> <host> * :<real name>"
        312     RPL_WHOISSERVER
                        "<nick> <server> :<server info>"
        313     RPL_WHOISOPERATOR
                        "<nick> :is an IRC operator"
        317     RPL_WHOISIDLE
                        "<nick> <integer> :seconds idle"
        318     RPL_ENDOFWHOIS
                        "<nick> :End of /WHOIS list"
        319     RPL_WHOISCHANNELS
                        "<nick> :{[@|+]<channel><space>}"

                - Replies 311 - 313, 317 - 319 are all replies
                  generated in response to a WHOIS message.  Given that
                  there are enough parameters present, the answering
                  server must either formulate a reply out of the above
                  numerics (if the query nick is found) or return an
                  error reply.  The '*' in RPL_WHOISUSER is there as
                  the literal character and not as a wild card.  For
                  each reply set, only RPL_WHOISCHANNELS may appear
                  more than once (for long lists of channel names).
                  The '@' and '+' characters next to the channel name
                  indicate whether a client is a channel operator or
                  has been granted permission to speak on a moderated
                  channel.  The RPL_ENDOFWHOIS reply is used to mark
                  the end of processing a WHOIS message.

        314     RPL_WHOWASUSER
                        "<nick> <user> <host> * :<real name>"
        369     RPL_ENDOFWHOWAS
                        "<nick> :End of WHOWAS"

                - When replying to a WHOWAS message, a server must use
                  the replies RPL_WHOWASUSER, RPL_WHOISSERVER or
                  ERR_WASNOSUCHNICK for each nickname in the presented
```

list. At the end of all reply batches, there must be RPL_ENDOFWHOWAS (even if there was only one reply and it was an error).

```
321     RPL_LISTSTART
                "Channel :Users  Name"
322     RPL_LIST
                "<channel> <# visible> :<topic>"
323     RPL_LISTEND
                ":End of /LIST"
```

- Replies RPL_LISTSTART, RPL_LIST, RPL_LISTEND mark the start, actual replies with data and end of the server's response to a LIST command. If there are no channels available to return, only the start and end reply must be sent.

```
324     RPL_CHANNELMODEIS
                "<channel> <mode> <mode params>"

331     RPL_NOTOPIC
                "<channel> :No topic is set"
332     RPL_TOPIC
                "<channel> :<topic>"
```

- When sending a TOPIC message to determine the channel topic, one of two replies is sent. If the topic is set, RPL_TOPIC is sent back else RPL_NOTOPIC.

```
341     RPL_INVITING
                "<channel> <nick>"
```

- Returned by the server to indicate that the attempted INVITE message was successful and is being passed onto the end client.

```
342     RPL_SUMMONING
                "<user> :Summoning user to IRC"
```

- Returned by a server answering a SUMMON message to indicate that it is summoning that user.

```
351     RPL_VERSION
                "<version>.<debuglevel> <server> :<comments>"
```

- Reply by the server showing its version details. The <version> is the version of the software being

used (including any patchlevel revisions) and the <debuglevel> is used to indicate if the server is running in "debug mode".

The "comments" field may contain any comments about the version or further version details.

```
352     RPL_WHOREPLY
                "<channel> <user> <host> <server> <nick> \
                 <H|G>[*][@|+] :<hopcount> <real name>"
315     RPL_ENDOFWHO
                "<name> :End of /WHO list"
```

- The RPL_WHOREPLY and RPL_ENDOFWHO pair are used to answer a WHO message. The RPL_WHOREPLY is only sent if there is an appropriate match to the WHO query. If there is a list of parameters supplied with a WHO message, a RPL_ENDOFWHO must be sent after processing each list item with <name> being the item.

```
353     RPL_NAMREPLY
                "<channel> :[[@|+]<nick> [[@|+]<nick> [...]]]"
366     RPL_ENDOFNAMES
                "<channel> :End of /NAMES list"
```

- To reply to a NAMES message, a reply pair consisting of RPL_NAMREPLY and RPL_ENDOFNAMES is sent by the server back to the client. If there is no channel found as in the query, then only RPL_ENDOFNAMES is returned. The exception to this is when a NAMES message is sent with no parameters and all visible channels and contents are sent back in a series of RPL_NAMEREPLY messages with a RPL_ENDOFNAMES to mark the end.

```
364     RPL_LINKS
                "<mask> <server> :<hopcount> <server info>"
365     RPL_ENDOFLINKS
                "<mask> :End of /LINKS list"
```

- In replying to the LINKS message, a server must send replies back using the RPL_LINKS numeric and mark the end of the list using an RPL_ENDOFLINKS reply.

```
367     RPL_BANLIST
                "<channel> <banid>"
368     RPL_ENDOFBANLIST
```

```
                "<channel> :End of channel ban list"

        - When listing the active 'bans' for a given channel,
          a server is required to send the list back using the
          RPL_BANLIST and RPL_ENDOFBANLIST messages.  A separate
          RPL_BANLIST is sent for each active banid.  After the
          banids have been listed (or if none present) a
          RPL_ENDOFBANLIST must be sent.

      371     RPL_INFO
                ":<string>"
      374     RPL_ENDOFINFO
                ":End of /INFO list"

        - A server responding to an INFO message is required to
          send all its 'info' in a series of RPL_INFO messages
          with a RPL_ENDOFINFO reply to indicate the end of the
          replies.

      375     RPL_MOTDSTART
                ":- <server> Message of the day - "
      372     RPL_MOTD
                ":- <text>"
      376     RPL_ENDOFMOTD
                ":End of /MOTD command"

        - When responding to the MOTD message and the MOTD file
          is found, the file is displayed line by line, with
          each line no longer than 80 characters, using
          RPL_MOTD format replies.  These should be surrounded
          by a RPL_MOTDSTART (before the RPL_MOTDs) and an
          RPL_ENDOFMOTD (after).

      381     RPL_YOUREOPER
                ":You are now an IRC operator"

        - RPL_YOUREOPER is sent back to a client which has
          just successfully issued an OPER message and gained
          operator status.

      382     RPL_REHASHING
                "<config file> :Rehashing"

        - If the REHASH option is used and an operator sends
          a REHASH message, an RPL_REHASHING is sent back to
          the operator.

      391     RPL_TIME
```

```
                    "<server> :<string showing server's local time>"

            - When replying to the TIME message, a server must send
              the reply using the RPL_TIME format above.  The string
              showing the time need only contain the correct day and
              time there.  There is no further requirement for the
              time string.

        392     RPL_USERSSTART
                    ":UserID   Terminal  Host"
        393     RPL_USERS
                    ":%-8s %-9s %-8s"
        394     RPL_ENDOFUSERS
                    ":End of users"
        395     RPL_NOUSERS
                    ":Nobody logged in"

            - If the USERS message is handled by a server, the
              replies RPL_USERSTART, RPL_USERS, RPL_ENDOFUSERS and
              RPL_NOUSERS are used.  RPL_USERSSTART must be sent
              first, following by either a sequence of RPL_USERS
              or a single RPL_NOUSER.  Following this is
              RPL_ENDOFUSERS.

        200     RPL_TRACELINK
                    "Link <version & debug level> <destination> \
                     <next server>"
        201     RPL_TRACECONNECTING
                    "Try. <class> <server>"
        202     RPL_TRACEHANDSHAKE
                    "H.S. <class> <server>"
        203     RPL_TRACEUNKNOWN
                    "???? <class> [<client IP address in dot form>]"
        204     RPL_TRACEOPERATOR
                    "Oper <class> <nick>"
        205     RPL_TRACEUSER
                    "User <class> <nick>"
        206     RPL_TRACESERVER
                    "Serv <class> <int>S <int>C <server> \
                     <nick!user|*!*>@<host|server>"
        208     RPL_TRACENEWTYPE
                    "<newtype> 0 <client name>"
        261     RPL_TRACELOG
                    "File <logfile> <debug level>"

            - The RPL_TRACE* are all returned by the server in
              response to the TRACE message.  How many are
              returned is dependent on the the TRACE message and
```

whether it was sent by an operator or not. There is no predefined order for which occurs first. Replies RPL_TRACEUNKNOWN, RPL_TRACECONNECTING and RPL_TRACEHANDSHAKE are all used for connections which have not been fully established and are either unknown, still attempting to connect or in the process of completing the 'server handshake'. RPL_TRACELINK is sent by any server which handles a TRACE message and has to pass it on to another server. The list of RPL_TRACELINKs sent in response to a TRACE command traversing the IRC network should reflect the actual connectivity of the servers themselves along that path. RPL_TRACENEWTYPE is to be used for any connection which does not fit in the other categories but is being displayed anyway.

```
211     RPL_STATSLINKINFO
                "<linkname> <sendq> <sent messages> \
                 <sent bytes> <received messages> \
                 <received bytes> <time open>"
212     RPL_STATSCOMMANDS
                "<command> <count>"
213     RPL_STATSCLINE
                "C <host> * <name> <port> <class>"
214     RPL_STATSNLINE
                "N <host> * <name> <port> <class>"
215     RPL_STATSILINE
                "I <host> * <host> <port> <class>"
216     RPL_STATSKLINE
                "K <host> * <username> <port> <class>"
218     RPL_STATSYLINE
                "Y <class> <ping frequency> <connect \
                 frequency> <max sendq>"
219     RPL_ENDOFSTATS
                "<stats letter> :End of /STATS report"
241     RPL_STATSLLINE
                "L <hostmask> * <servername> <maxdepth>"
242     RPL_STATSUPTIME
                ":Server Up %d days %d:%02d:%02d"
243     RPL_STATSOLINE
                "O <hostmask> * <name>"
244     RPL_STATSHLINE
                "H <hostmask> * <servername>"

221     RPL_UMODEIS
                "<user mode string>"
```

```
                - To answer a query about a client's own mode,
                  RPL_UMODEIS is sent back.

        251     RPL_LUSERCLIENT
                        ":There are <integer> users and <integer> \
                         invisible on <integer> servers"
        252     RPL_LUSEROP
                        "<integer> :operator(s) online"
        253     RPL_LUSERUNKNOWN
                        "<integer> :unknown connection(s)"
        254     RPL_LUSERCHANNELS
                        "<integer> :channels formed"
        255     RPL_LUSERME
                        ":I have <integer> clients and <integer> \
                          servers"

                - In processing an LUSERS message, the server
                  sends a set of replies from RPL_LUSERCLIENT,
                  RPL_LUSEROP, RPL_USERUNKNOWN,
                  RPL_LUSERCHANNELS and RPL_LUSERME.  When
                  replying, a server must send back
                  RPL_LUSERCLIENT and RPL_LUSERME.  The other
                  replies are only sent back if a non-zero count
                  is found for them.

        256     RPL_ADMINME
                        "<server> :Administrative info"
        257     RPL_ADMINLOC1
                        ":<admin info>"
        258     RPL_ADMINLOC2
                        ":<admin info>"
        259     RPL_ADMINEMAIL
                        ":<admin info>"

                - When replying to an ADMIN message, a server
                  is expected to use replies RLP_ADMINME
                  through to RPL_ADMINEMAIL and provide a text
                  message with each.  For RPL_ADMINLOC1 a
                  description of what city, state and country
                  the server is in is expected, followed by
                  details of the university and department
                  (RPL_ADMINLOC2) and finally the administrative
                  contact for the server (an email address here
                  is required) in RPL_ADMINEMAIL.
```

### 6.3 Reserved numerics.

These numerics are not described above since they fall into one of the following categories:

1. no longer in use;

2. reserved for future planned use;

3. in current use but are part of a non-generic 'feature' of the current IRC server.

| | | | |
|---|---|---|---|
| 209 | RPL_TRACECLASS | 217 | RPL_STATSQLINE |
| 231 | RPL_SERVICEINFO | 232 | RPL_ENDOFSERVICES |
| 233 | RPL_SERVICE | 234 | RPL_SERVLIST |
| 235 | RPL_SERVLISTEND | | |
| 316 | RPL_WHOISCHANOP | 361 | RPL_KILLDONE |
| 362 | RPL_CLOSING | 363 | RPL_CLOSEEND |
| 373 | RPL_INFOSTART | 384 | RPL_MYPORTIS |
| 466 | ERR_YOUWILLBEBANNED | 476 | ERR_BADCHANMASK |
| 492 | ERR_NOSERVICEHOST | | |

## 7. Client and server authentication

Clients and servers 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). These checks are possible on all connections although the password check is only commonly used with servers.

An additional check that is becoming of more and more common is that of the username responsible for making the connection. Finding the username of the other end of the connection typically involves connecting to an authentication server such as IDENT as described in RFC 1413.

Given that without passwords it is not easy to reliably determine who is on the other end of a network connection, use of passwords is strongly recommended on inter-server connections in addition to any other measures such as using an ident server.

## 8. Current implementations

The only current implementation of this protocol is the IRC server, version 2.8. Earlier versions may implement some or all of the commands described by this document with NOTICE messages replacing

many of the numeric replies. Unfortunately, due to backward compatibility requirements, the implementation of some parts of this document varies with what is laid out. On notable difference is:

* recognition that any LF or CR anywhere in a message marks the end of that message (instead of requiring CR-LF);

The rest of this section deals with issues that are mostly of importance to those who wish to implement a server but some parts also apply directly to clients as well.

### 8.1 Network protocol: TCP - why it is best used here.

IRC has been implemented on top of TCP since TCP supplies a reliable network protocol which is well suited to this scale of conferencing. The use of multicast IP is an alternative, but it is not widely available or supported at the present time.

#### 8.1.1 Support of Unix sockets

Given that Unix domain sockets allow listen/connect operations, the current implementation can be configured to listen and accept both client and server connections on a Unix domain socket. These are recognized as sockets where the hostname starts with a '/'.

When providing any information about the connections on a Unix domain socket, the server is required to supplant the actual hostname in place of the pathname unless the actual socket name is being asked for.

### 8.2 Command Parsing

To provide useful 'non-buffered' network IO for clients and servers, each connection is given its own private 'input buffer' in which the results of the most recent read and parsing are kept. A buffer size of 512 bytes is used so as to hold 1 full message, although, this will usually hold several commands. The private buffer is parsed after every read operation for valid messages. When dealing with multiple messages from one client in the buffer, care should be taken in case one happens to cause the client to be 'removed'.

### 8.3 Message delivery

It is common to find network links saturated or hosts to which you are sending data unable to send data. Although Unix typically handles this through the TCP window and internal buffers, the server often has large amounts of data to send (especially when a new server-server link forms) and the small buffers provided in the

kernel are not enough for the outgoing queue. To alleviate this problem, a "send queue" is used as a FIFO queue for data to be sent. A typical "send queue" may grow to 200 Kbytes on a large IRC network with a slow network connection when a new server connects.

When polling its connections, a server will first read and parse all incoming data, queuing any data to be sent out. When all available input is processed, the queued data is sent. This reduces the number of write() system calls and helps TCP make bigger packets.

### 8.4 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 using the appropriate procedures. A connection is also dropped if its sendq grows beyond the maximum allowed, because it is better to close a slow connection than have a server process block.

### 8.5 Establishing a server to client connection

Upon connecting to an IRC server, a client is sent the MOTD (if present) as well as the current user/server count (as per the LUSER command). The server is also required to give an unambiguous message to the client which states its name and version as well as any other introductory messages which may be deemed appropriate.

After dealing with this, the server must then send out the new user's nickname and other information as supplied by itself (USER command) and as the server could discover (from DNS/authentication servers). The server must send this information out with NICK first followed by USER.

### 8.6 Establishing a server-server connection.

The process of establishing of a server-to-server connection is fraught with danger since there are many possible areas where problems can occur - the least of which are race conditions.

After a server has received a connection following by a PASS/SERVER pair which were recognised as being valid, the server should then reply with its own PASS/SERVER information for that connection as well as all of the other state information it knows about as described below.

When the initiating server receives a PASS/SERVER pair, it too then

checks that the server responding is authenticated properly before accepting the connection to be that server.

### 8.6.1 Server exchange of state information when connecting

The order of state information being exchanged between servers is essential. The required order is as follows:

* all known other servers;

* all known user information;

* all known channel information.

Information regarding servers is sent via extra SERVER messages, user information with NICK/USER/MODE/JOIN messages and channels with MODE messages.

NOTE: channel topics are *NOT* exchanged here because the TOPIC command overwrites any old topic information, so at best, the two sides of the connection would exchange topics.

By passing the state information about servers first, any collisions with servers that already exist occur before nickname collisions due to a second server introducing a particular nickname. Due to the IRC network only being able to exist as an acyclic graph, it may be possible that the network has already reconnected in another location, the place where the collision occurs indicating where the net needs to split.

## 8.7 Terminating server-client connections

When a client connection closes, a QUIT message is generated on behalf of the client by the server to which the client connected. No other message is to be generated or used.

## 8.8 Terminating server-server connections

If a server-server connection is closed, either via a remotely generated SQUIT or 'natural' causes, the rest of the connected IRC network must have its information updated with by the server which detected the closure. The server then sends a list of SQUITs (one for each server behind that connection) and a list of QUITs (again, one for each client behind that connection).

### 8.9 Tracking nickname changes

All IRC servers are required to keep a history of recent nickname changes. This is required to allow the server to have a chance of keeping in touch of things when nick-change race conditions occur with commands which manipulate them. Commands which must trace nick changes are:

* KILL (the nick being killed)

* MODE (+/- o,v)

* KICK (the nick being kicked)

No other commands are to have nick changes checked for.

In the above cases, the server is required to first check for the existence of the nickname, then check its history to see who that nick currently belongs to (if anyone!). This reduces the chances of race conditions but they can still occur with the server ending up affecting the wrong client. When performing a change trace for an above command it is recommended that a time range be given and entries which are too old ignored.

For a reasonable history, a server should be able to keep previous nickname for every client it knows about if they all decided to change. This size is limited by other factors (such as memory, etc).

### 8.10 Flood control of clients

With a large network of interconnected IRC servers, it is quite easy for any single client attached to the network to supply a continuous stream of messages that result in not only flooding the network, but also degrading the level of service provided to others. Rather than require every 'victim' to be provide their own protection, flood protection was written into the server and is applied to all clients except services. The current algorithm is as follows:

* check to see if client's `message timer' is less than current time (set to be equal if it is);

* read any data present from the client;

* while the timer is less than ten seconds ahead of the current time, parse any present messages and penalize the client by 2 seconds for each message;

which in essence means that the client may send 1 message every 2

seconds without being adversely affected.

### 8.11 Non-blocking lookups

In a real-time environment, it is essential that a server process do as little waiting as possible so that all the clients are serviced fairly. Obviously this requires non-blocking IO on all network read/write operations. For normal server connections, this was not difficult, but there are other support operations that may cause the server to block (such as disk reads). Where possible, such activity should be performed with a short timeout.

#### 8.11.1 Hostname (DNS) lookups

Using the standard resolver libraries from Berkeley and others has meant large delays in some cases where replies have timed out. To avoid this, a separate set of DNS routines were written which were setup for non-blocking IO operations and then polled from within the main server IO loop.

#### 8.11.2 Username (Ident) lookups

Although there are numerous ident libraries for use and inclusion into other programs, these caused problems since they operated in a synchronous manner and resulted in frequent delays. Again the solution was to write a set of routines which would cooperate with the rest of the server and work using non-blocking IO.

### 8.12 Configuration File

To provide a flexible way of setting up and running the server, it is recommended that a configuration file be used which contains instructions to the server on the following:

* which hosts to accept client connections from;

* which hosts to allow to connect as servers;

* which hosts to connect to (both actively and passively);

* information about where the server is (university, city/state, company are examples of this);

* who is responsible for the server and an email address at which they can be contacted;

* hostnames and passwords for clients which wish to be given

access to restricted operator commands.

In specifying hostnames, both domain names and use of the 'dot' notation (127.0.0.1) should both be accepted. It must be possible to specify the password to be used/accepted for all outgoing and incoming connections (although the only outgoing connections are those to other servers).

The above list is the minimum requirement for any server which wishes to make a connection with another server. Other items which may be of use are:

* specifying which servers other server may introduce;

* how deep a server branch is allowed to become;

* hours during which clients may connect.

### 8.12.1 Allowing clients to connect

A server should use some sort of 'access control list' (either in the configuration file or elsewhere) that is read at startup and used to decide what hosts clients may use to connect to it.

Both 'deny' and 'allow' should be implemented to provide the required flexibility for host access control.

### 8.12.2 Operators

The granting of operator privileges to a disruptive person can have dire consequences for the well-being of the IRC net in general due to the powers given to them. Thus, the acquisition of such powers should not be very easy. The current setup requires two 'passwords' to be used although one of them is usually easy guessed. Storage of oper passwords in configuration files is preferable to hard coding them in and should be stored in a crypted format (ie using crypt(3) from Unix) to prevent easy theft.

### 8.12.3 Allowing servers to connect

The interconnection of server is not a trivial matter: a bad connection can have a large impact on the usefulness of IRC. Thus, each server should have a list of servers to which it may connect and which servers may connect to it. Under no circumstances should a server allow an arbitrary host to connect as a server. In addition to which servers may and may not connect, the configuration file should also store the password and other characteristics of that link.

### 8.12.4 Administrivia

To provide accurate and valid replies to the ADMIN command (see section 4.3.7), the server should find the relevant details in the configuration.

### 8.13 Channel membership

The current server allows any registered local user to join upto 10 different channels. There is no limit imposed on non-local users so that the server remains (reasonably) consistant with all others on a channel membership basis

## 9. Current problems

There are a number of recognized problems with this protocol, all of which hope to be solved sometime in the near future during its rewrite. Currently, work is underway to find working solutions to these problems.

### 9.1 Scalability

It is widely recognized that this protocol does not scale sufficiently well when used in a large arena. The main problem comes from the requirement that all servers know about all other servers and users and that information regarding them be updated as soon as it changes. It is also desirable to keep the number of servers low so that the path length between any two points is kept minimal and the spanning tree as strongly branched as possible.

### 9.2 Labels

The current IRC protocol has 3 types of labels: the nickname, the channel name and the server name. Each of the three types has its own domain and no duplicates are allowed inside that domain. Currently, it is possible for users to pick the label for any of the three, resulting in collisions. It is widely recognized that this needs reworking, with a plan for unique names for channels and nicks that don't collide being desirable as well as a solution allowing a cyclic tree.

#### 9.2.1 Nicknames

The idea of the nickname on IRC is very convenient for users to use when talking to each other outside of a channel, but there is only a finite nickname space and being what they are, its not uncommon for several people to want to use the same nick. If a nickname is chosen by two people using this protocol, either one will not succeed or

both will removed by use of KILL (4.6.1).

### 9.2.2 Channels

The current channel layout requires that all servers know about all channels, their inhabitants and properties. Besides not scaling well, the issue of privacy is also a concern. A collision of channels is treated as an inclusive event (both people who create the new channel are considered to be members of it) rather than an exclusive one such as used to solve nickname collisions.

### 9.2.3 Servers

Although the number of servers is usually small relative to the number of users and channels, they two currently required to be known globally, either each one separately or hidden behind a mask.

## 9.3 Algorithms

In some places within the server code, it has not been possible to avoid N^2 algorithms such as checking the channel list of a set of clients.

In current server versions, there are no database consistency checks, each server assumes that a neighbouring server is correct. This opens the door to large problems if a connecting server is buggy or otherwise tries to introduce contradictions to the existing net.

Currently, because of the lack of unique internal and global labels, there are a multitude of race conditions that exist. These race conditions generally arise from the problem of it taking time for messages to traverse and effect the IRC network. Even by changing to unique labels, there are problems with channel-related commands being disrupted.

# 10. Current support and availability

Mailing lists for IRC related discussion:
Future protocol: ircd-three-request@eff.org
General discussion: operlist-request@eff.org

Software implemenations
cs.bu.edu:/irc
nic.funet.fi:/pub/irc
coombs.anu.edu.au:/pub/irc

Newsgroup: alt.irc

Security Considerations

Security issues are discussed in sections 4.1, 4.1.1, 4.1.3, 5.5, and 7.

## 12. Authors' Addresses

Jarkko Oikarinen
Tuirantie 17 as 9
90500 OULU
FINLAND

Email: jto@tolsun.oulu.fi

Darren Reed
4 Pateman Street
Watsonia, Victoria 3087
Australia

Email: avalon@coombs.anu.edu.au