Network Working Group
Request for Comments: 2813
Updates: 1459
Category: Informational

C. Kalt
April 2000

Internet Relay Chat: Server Protocol

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Abstract

While based on the client-server model, the IRC (Internet Relay Chat) protocol allows servers to connect to each other effectively forming a network.

This document defines the protocol used by servers to talk to each other. It was originally a superset of the client protocol but has evolved differently.

First formally documented in May 1993 as part of RFC 1459 [IRC], most of the changes brought since then can be found in this document as development was focused on making the protocol scale better. Better scalability has allowed existing world-wide networks to keep growing and reach sizes which defy the old specification.

Table of Contents

1. Introduction

This document is intended for people working on implementing an IRC server but will also be useful to anyone implementing an IRC service.

Servers provide the three basic services required for realtime conferencing defined by the "Internet Relay Chat: Architecture" [IRC-ARCH]: client locator (via the client protocol [IRC-CLIENT]), message relaying (via the server protocol defined in this document) and channel hosting and management (following specific rules [IRC-CHAN]).

2. Global database

Although the IRC Protocol defines a fairly distributed model, each server maintains a "global state database" about the whole IRC network. This database is, in theory, identical on all servers.

2.1 Servers

Servers are uniquely identified by their name which has a maximum length of sixty three (63) characters. See the protocol grammar rules (section 3.3.1) for what may and may not be used in a server name.

Each server is typically known by all other servers, however it is possible to define a "hostmask" to group servers together according to their name. Inside the hostmasked area, all the servers have a name which matches the hostmask, and any other server with a name matching the hostmask SHALL NOT be connected to the IRC network outside the hostmasked area. Servers which are outside the area have no knowledge of the individual servers present inside the area, instead they are presented with a virtual server which has the hostmask for name.

2.2 Clients

For each client, all servers MUST have the following information: a netwide unique identifier (whose format depends on the type of client) and the server to which the client is connected.

2.2.1 Users

Each user is distinguished from other users by a unique nickname having a maximum length of nine (9) characters. See the protocol grammar rules (section 3.3.1) 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 users: the name of the host that the user is running on, the username of the user on that host, and the server to which the client is connected.

2.2.2 Services

Each service is distinguished from other services by a service name composed of a nickname and a server name. The nickname has a maximum length of nine (9) characters. See the protocol grammar rules (section 3.3.1) for what may and may not be used in a nickname. The server name used to compose the service name is the name of the server to which the service is connected. In addition to this service name all servers MUST know the service type.

Services differ from users by the format of their identifier, but more importantly services and users don’t have the same type of access to the server: services can request part or all of the global state information that a server maintains, but have a more restricted set of commands available to them (See "IRC Client Protocol" [IRC-CLIENT] for details on which) and are not allowed to join channels. Finally services are not usually subject to the "Flood control" mechanism described in section 5.8.

2.3 Channels

Alike services, channels have a scope [IRC-CHAN] and are not necessarily known to all servers. When a channel existence is known to a server, the server MUST keep track of the channel members, as well as the channel modes.

3. The IRC Server Specification

3.1 Overview

The protocol as described herein is for use with server to server connections. For client to server connections, see the IRC Client Protocol specification.

There are, however, more restrictions on client connections (which are considered to be untrustworthy) than on server connections.

3.2 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 US-ASCII terminal and a telnet connection.

Because of IRC’s Scandinavian 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, or channel names.

3.3 Messages

Servers and clients send each other messages which may or may not generate a reply. Most communication between servers do not generate any reply, as servers mostly perform routing tasks for the clients.

Each IRC message may consist of up to three main parts: the prefix (OPTIONAL), the command, and the command parameters (maximum of fifteen (15)). The prefix, command, and all parameters are separated by one ASCII space character (0x20) each.

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 a prefix when sending a message from themselves; if they use one, the only valid prefix is the registered nickname associated with the client.

When a server receives a message, it MUST identify its source using the (eventually assumed) prefix. If the prefix cannot be found in the server's internal database, it MUST be discarded, and if the prefix indicates the message comes from an (unknown) server, the link from which the message was received MUST be dropped. Dropping a link in such circumstances is a little excessive but necessary to maintain the integrity of the network and to prevent future problems. Another common error condition is that the prefix found in the server's internal database identifies a different source (typically a source registered from a different link than from which the message arrived). If the message was received from a server link and the prefix identifies a client, a KILL message MUST be issued for the client and sent to all servers. In other cases, the link from which the message arrived SHOULD be dropped for clients, and MUST be dropped for servers. In all cases, the message MUST be discarded.

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 5 for more details about current implementations.

3.3.1 Message format in Augmented 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 (<params>).

The Augmented BNF representation for this is found in "IRC Client Protocol" [IRC-CLIENT].

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.

3.4 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 "IRC Client Protocol" [IRC-CLIENT].

4. Message Details

All the messages recognized by the IRC server and client are described in the IRC Client Protocol specification.

Where the reply ERR_NOSUCHSERVER is returned, it means that the target of the message could not be found. The server MUST NOT send any other replies after this error 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 follow from incorrect command, a destination which is otherwise unknown to the server (server, client 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.

The message details for client to server communication are described in the "IRC Client Protocol" [IRC-CLIENT]. Some sections in the following pages apply to some of these messages, they are additions to the message specifications which are only relevant to server to server communication, or to the server implementation. The messages which are introduced here are only used for server to server communication.

4.1 Connection Registration

The commands described here are used to register a connection with another IRC server.

4.1.1 Password message

```
   Command: PASS
Parameters: <password> <version> <flags> [<options>]
```

The PASS command is used to set a 'connection password'. The password MUST be set before any attempt to register the connection is made. Currently this means that servers MUST send a PASS command before any SERVER command. Only one (1) PASS command SHALL be accepted from a connection.

The last three (3) parameters MUST be ignored if received from a client (e.g. a user or a service). They are only relevant when received from a server.

The <version> parameter is a string of at least four (4) characters, and up to fourteen (14) characters. The first four (4) characters MUST be digits and indicate the protocol version known by the server issuing the message. The protocol described by this document is version 2.10 which is encoded as "0210". The remaining OPTIONAL characters are implementation dependent and should describe the software version number.

The <flags> parameter is a string of up to one hundred (100) characters. It is composed of two substrings separated by the character "|" (%x7C). If present, the first substring MUST be the name of the implementation. The reference implementation (See Section 8, "Current support and availability") uses the string "IRC". If a different implementation is written, which needs an identifier, then that identifier should be registered through publication of an RFC. The second substring is implementation dependent. Both substrings are OPTIONAL, but the character "|" is REQUIRED. The character "|" MUST NOT appear in either substring.

Finally, the last parameter, <options>, is used for link options. The only options defined by the protocol are link compression (using the character "Z"), and an abuse protection flag (using the character

"P"). See sections 5.3.1.1 (Compressed server to server links) and 5.3.1.2 (Anti abuse protections) respectively for more information on these options.

Numeric Replies:

```
        ERR_NEEDMOREPARAMS              ERR_ALREADYREGISTRED
```

Example:

```
     PASS moresecretpassword 0210010000 IRC|aBgH$ Z
```

4.1.2 Server message

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

The SERVER command is used to register a new server. A new connection introduces itself as a server to its peer. This message is also used to pass server data over whole net. When a new server is connected to net, information about it MUST be broadcasted to the whole network.

The <info> parameter may contain space characters.

<hopcount> is used to give all servers some internal information on how far away each server is. Local peers have a value of 0, and each passed server increments the value. 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 <token> parameter is an unsigned number used by servers as an identifier. This identifier is subsequently used to reference a server in the NICK and SERVICE messages sent between servers. Server tokens only have a meaning for the point-to-point peering they are used and MUST be unique for that connection. They are not global.

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). Because of the severity of such event, error replies are usually sent using the "ERROR" command rather than a numeric.

If a SERVER message is parsed and it attempts to introduce a server which is already known to the receiving server, the connection, from which that message arrived, MUST be closed (following the correct procedures), since a duplicate route to a server has been formed and the acyclic nature of the IRC tree breaks. In some conditions, the connection from which the already known server has registered MAY be closed instead. It should be noted that this kind of error can also be the result of a second running server, problem which cannot be fixed within the protocol and typically requires human intervention. This type of problem is particularly insidious, as it can quite easily result in part of the IRC network to be isolated, with one of the two servers connected to each partition therefore making it impossible for the two parts to unite.

Numeric Replies:

ERR_ALREADYREGISTRED

Example:

```
SERVER test.oulu.fi 1 1 :Experimental server ; New server
                                test.oulu.fi introducing itself and
                                attempting to register.

:tolsun.oulu.fi SERVER csd.bu.edu 5 34 :BU Central Server ; Server
                                tolsun.oulu.fi is our uplink for
                                csd.bu.edu which is 5 hops away.  The
                                token "34" will be used by
                                tolsun.oulu.fi when introducing new
                                users or services connected to
                                csd.bu.edu.
```

4.1.3 Nick

```
   Command: NICK
Parameters: <nickname> <hopcount> <username> <host> <servertoken>
            <umode> <realname>
```

This form of the NICK message MUST NOT be allowed from user connections. However, it MUST be used instead of the NICK/USER pair to notify other servers of new users joining the IRC network.

This message is really the combination of three distinct messages: NICK, USER and MODE [IRC-CLIENT].

The <hopcount> parameter is used by servers to indicate how far away a user is from its home server. A local connection has a hopcount of 0. The hopcount value is incremented by each passed server.

The <servertoken> parameter replaces the <servername> parameter of the USER (See section 4.1.2 for more information on server tokens).

Examples:

```
NICK syrk 5 kalt millennium.stealth.net 34 +i :Christophe Kalt ; New
                                user with nickname "syrk", username
                                "kalt", connected from host
                                "millennium.stealth.net" to server
                                "34" ("csd.bu.edu" according to the
                                previous example).

:krys NICK syrk                 ; The other form of the NICK message,
                                as defined in "IRC Client Protocol"
                                [IRC-CLIENT] and used between
                                servers: krys changed his nickname to
                                syrk
```

4.1.4 Service message

```
   Command: SERVICE
Parameters: <servicename> <servertoken> <distribution> <type>
             <hopcount> <info>
```

The SERVICE command is used to introduce a new service. This form of the SERVICE message SHOULD NOT be allowed from client (unregistered, or registered) connections. However, it MUST be used between servers to notify other servers of new services joining the IRC network.

The <servertoken> is used to identify the server to which the service is connected. (See section 4.1.2 for more information on server tokens).

The <hopcount> parameter is used by servers to indicate how far away a service is from its home server. A local connection has a hopcount of 0. The hopcount value is incremented by each passed server.

The <distribution> parameter is used to specify the visibility of a service. The service may only be known to servers which have a name matching the distribution. For a matching server to have knowledge of the service, the network path between that server and the server to which the service is connected MUST be composed of servers whose names all match the mask. Plain "*" is used when no restriction is wished.

The <type> parameter is currently reserved for future usage.

Numeric Replies:

```
        ERR_ALREADYREGISTRED            ERR_NEEDMOREPARAMS
        ERR_ERRONEUSNICKNAME
        RPL_YOURESERVICE                RPL_YOURHOST
        RPL_MYINFO
```

Example:

```
SERVICE dict@irc.fr 9 *.fr 0 1 :French Dictionary r" registered on
                                   server "9" is being announced to
                                   another server.  This service will
                                   only be available on servers whose
                                   name matches "*.fr".
```

4.1.5 Quit

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

A client session ends 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 or service name.

When "netsplit" (See Section 4.1.6) 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 either that of the server which has become disconnected or that of the server to which the leaving client was connected:

```
<Quit Message> =  ":" servername SPACE servername
```

Because the "Quit Message" has a special meaning for "netsplits", servers SHOULD NOT allow a client to use a <Quit Message> in the format described above.

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. Typically, this is done by reporting a system specific error.

Numeric Replies:

None.

Examples:

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

4.1.6 Server quit message

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

The SQUIT message has two distinct uses.

The first one (described in "Internet Relay Chat: Client Protocol" [IRC-CLIENT]) allows operators to break a local or remote server link. This form of the message is also eventually used by servers to break a remote server link.

The second use of this message is needed to inform other servers when a "network split" (also known as "netsplit") occurs, in other words to inform other servers 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 name of the other server as the server parameter, which then closes its connection to the quitting server.

The <comment> is filled in by servers which SHOULD place an error or similar message here.

Both of the servers which are on either side of the connection being closed are REQUIRED 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 MAY be sent to the other still connected servers on behalf of all clients behind that quitting 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. Messages to channel members are generated by each client’s local server.

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.

When a client is removed as the result of a SQUIT message, the server SHOULD add the nickname to the list of temporarily unavailable nicknames in an attempt to prevent future nickname collisions. See section 5.7 (Tracking recently used nicknames) for more information on this procedure.

Numeric replies:

```
        ERR_NOPRIVILEGES                ERR_NOSUCHSERVER
        ERR_NEEDMOREPARAMS
```

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 users). In implementing these, a number of race conditions are inevitable when users 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>[ %x7 <modes> ]
            *( "," <channel>[ %x7 <modes> ] )
```

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 local server the client is connected to; all other servers automatically add the user to the channel when the command is received from other servers.

Optionally, the user status (channel modes ’O’, ’o’, and ’v’) on the channel may be appended to the channel name using a control G (^G or ASCII 7) as separator. Such data MUST be ignored if the message wasn’t received from a server. This format MUST NOT be sent to clients, it can only be used between servers and SHOULD be avoided.

The JOIN command MUST 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 and NOTICE messages to the channel.

Numeric Replies:

```
        ERR_NEEDMOREPARAMS              ERR_BANNEDFROMCHAN
        ERR_INVITEONLYCHAN              ERR_BADCHANNELKEY
        ERR_CHANNELISFULL               ERR_BADCHANMASK
        ERR_NOSUCHCHANNEL               ERR_TOOMANYCHANNELS
        ERR_TOOMANYTARGETS              ERR_UNAVAILRESOURCE
        RPL_TOPIC
```

Examples:

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

4.2.2 Njoin message

```
   Command: NJOIN
Parameters: <channel> [ "@@" / "@" ] [ "+" ] <nickname>
                      *( "," [ "@@" / "@" ] [ "+" ] <nickname> )
```

The NJOIN message is used between servers only. If such a message is received from a client, it MUST be ignored. It is used when two servers connect to each other to exchange the list of channel members for each channel.

Even though the same function can be performed by using a succession of JOIN, this message SHOULD be used instead as it is more efficient. The prefix "@@" indicates that the user is the "channel creator", the character "@" alone indicates a "channel operator", and the character '+' indicates that the user has the voice privilege.

Numeric Replies:

```
        ERR_NEEDMOREPARAMS              ERR_NOSUCHCHANNEL
        ERR_ALREADYREGISTRED
```

Examples:

```
:ircd.stealth.net NJOIN #Twilight_zone :@WiZ,+syrk,avalon ; NJOIN
                                message from ircd.stealth.net
                                announcing users joining the
                                #Twilight_zone channel: WiZ with
                                channel operator status, syrk with
                                voice privilege and avalon with no
                                privilege.
```

4.2.3 Mode message

The MODE message is a dual-purpose command in IRC. It allows both usernames and channels to have their mode changed.

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

It is REQUIRED that servers are able to change channel modes so that "channel creator" and "channel operators" may be created.

5. Implementation details

A the time of writing, the only current implementation of this protocol is the IRC server, version 2.10. 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. One 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.

5.1 Connection ’Liveness’

To detect when a connection has died or become unresponsive, the server MUST poll each of its connections. The PING command (See "IRC Client Protocol" [IRC-CLIENT]) is used if the server doesn’t get a response from its peer in a given amount of time.

If a connection doesn’t respond in time, its connection is closed using the appropriate procedures.

5.2 Accepting a client to server connection

5.2.1 Users

When a server successfully registers a new user connection, it is REQUIRED to send to the user unambiguous messages stating: the user identifiers upon which it was registered (RPL_WELCOME), the server name and version (RPL_YOURHOST), the server birth information (RPL_CREATED), available user and channel modes (RPL_MYINFO), and it MAY send any introductory messages which may be deemed appropriate.

In particular the server SHALL send the current user/service/server count (as per the LUSER reply) and finally the MOTD (if any, as per the MOTD reply).

After dealing with registration, the server MUST then send out to other servers the new user’s nickname (NICK message), other information as supplied by itself (USER message) and as the server could discover (from DNS servers). The server MUST NOT send this information out with a pair of NICK and USER messages as defined in "IRC Client Protocol" [IRC-CLIENT], but MUST instead take advantage of the extended NICK message defined in section 4.1.3.

5.2.2 Services

Upon successfully registering a new service connection, the server is subject to the same kind of REQUIREMENTS as for a user. Services being somewhat different, only the following replies are sent: RPL_YOURESERVICE, RPL_YOURHOST, RPL_MYINFO.

After dealing with this, the server MUST then send out to other servers (SERVICE message) the new service’s nickname and other information as supplied by the service (SERVICE message) and as the server could discover (from DNS servers).

5.3 Establishing a server-server connection.

The process of establishing 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 recognized 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.

5.3.1 Link options

Server links are based on a common protocol (defined by this document) but a particular link MAY set specific options using the PASS message (See Section 4.1.1).

5.3.1.1 Compressed server to server links

If a server wishes to establish a compressed link with its peer, it MUST set the 'Z' flag in the options parameter to the PASS message. If both servers request compression and both servers are able to initialize the two compressed streams, then the remainder of the communication is to be compressed. If any server fails to initialize the stream, it will send an uncompressed ERROR message to its peer and close the connection.

The data format used for the compression is described by RFC 1950 [ZLIB], RFC 1951 [DEFLATE] and RFC 1952 [GZIP].

5.3.1.2 Anti abuse protections

Most servers implement various kinds of protections against possible abusive behaviours from non trusted parties (typically users). On some networks, such protections are indispensable, on others they are superfluous. To require that all servers implement and enable such features on a particular network, the 'P' flag is used when two servers connect. If this flag is present, it means that the server protections are enabled, and that the server REQUIRES all its server links to enable them as well.

Commonly found protections are described in sections 5.7 (Tracking recently used nicknames) and 5.8 (Flood control of clients).

5.3.2 State information exchange when connecting

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

* all known servers;

* all known client information;

* all known channel information.

Information regarding servers is sent via extra SERVER messages, client information with NICK and SERVICE messages and channels with NJOIN/MODE messages.

NOTE: channel topics SHOULD NOT be 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 caused by 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. In this event, the place where the server collision occurs indicates where the net needs to split.

5.4 Terminating server-client connections

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

5.5 Terminating server-server connections

If a server-server connection is closed, either via a SQUIT command or "natural" causes, the rest of the connected IRC network MUST have its information updated by the server which detected the closure. The terminating server then sends a list of SQUITs (one for each server behind that connection). (See Section 4.1.6 (SQUIT)).

5.6 Tracking nickname changes

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

* KILL (the nick being disconnected)

* MODE (+/- o,v on channels)

* KICK (the nick being removed from channel)

No other commands need to check nick changes.

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 now 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).

5.7 Tracking recently used nicknames

This mechanism is commonly known as "Nickname Delay", it has been proven to significantly reduce the number of nickname collisions resulting from "network splits"/reconnections as well as abuse.

In addition of keeping track of nickname changes, servers SHOULD keep track of nicknames which were recently used and were released as the result of a "network split" or a KILL message. These nicknames are then unavailable to the server local clients and cannot be re-used (even though they are not currently in use) for a certain period of time.

The duration for which a nickname remains unavailable SHOULD be set considering many factors among which are the size (user wise) of the IRC network, and the usual duration of "network splits". It SHOULD be uniform on all servers for a given IRC network.

5.8 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 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 (10) seconds ahead of the current time, parse any present messages and penalize the client by two (2) seconds for each message;

* additional penalties MAY be used for specific commands which generate a lot of traffic across the network.

This in essence means that the client may send one (1) message every two (2) seconds without being adversely affected. Services MAY also be subject to this mechanism.

5.9 Non-blocking lookups

In a real-time environment, it is essential that a server process does 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.

5.9.1 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 for the current implementation. Routines were setup for non-blocking IO operations with local cache, and then polled from within the main server IO loop.

5.9.2 Username (Ident) lookups

Although there are numerous ident libraries (implementing the "Identification Protocol" [IDENT]) 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.

6. Current problems

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

6.1 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 clients 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.

6.2 Labels

The current IRC protocol has 4 types of labels: the nickname, the channel name, the server name and the service name. Each of the four 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 first three, resulting in collisions. It is widely recognized that this needs reworking, with a plan for unique names for nicks that don’t collide being desirable as well as a solution allowing a cyclic tree.

6.2.1 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, it’s 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 be removed by use of KILL (See Section 3.7.1 of "IRC Client Protocol" [IRC-CLIENT]).

6.2.2 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 (people from both nets on channel with common name are considered to be members of it) rather than an exclusive one such as used to solve nickname collisions.

This protocol defines "Safe Channels" which are very unlikely to be the subject of a channel collision. Other channel types are kept for backward compatibility.

6.2.3 Servers

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

6.3 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 only few database consistency checks, most of the time 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.

7. Security Considerations

7.1 Authentication

Servers only have two means of authenticating incoming connections: plain text password, and DNS lookups. While these methods are weak and widely recognized as unsafe, their combination has proven to be sufficient in the past:

* public networks typically allow user connections with only few restrictions, without requiring accurate authentication.

* private networks which operate in a controlled environment often use home-grown authentication mechanisms not available on the internet: reliable ident servers [IDENT], or other proprietary mechanisms.

The same comments apply to the authentication of IRC Operators.

It should also be noted that while there has been no real demand over the years for stronger authentication, and no real effort to provide better means to safely authenticate users, the current protocol offers enough to be able to easily plug-in external authentication methods based on the information that a client can submit to the server upon connection: nickname, username, password.

7.2 Integrity

Since the PASS and OPER messages of the IRC protocol are sent in clear text, a stream layer encryption mechanism (like "The TLS Protocol" [TLS]) could be used to protect these transactions.

8. Current support and availability

Mailing lists for IRC related discussion:
General discussion: ircd-users@irc.org
Protocol development: ircd-dev@irc.org

Software implementations:
ftp://ftp.irc.org/irc/server
ftp://ftp.funet.fi/pub/unix/irc
ftp://coombs.anu.edu.au/pub/irc

Newsgroup: alt.irc

9. Acknowledgements

Parts of this document were copied from the RFC 1459 [IRC] which first formally documented the IRC Protocol. It has also benefited from many rounds of review and comments. In particular, the following people have made significant contributions to this document:

Matthew Green, Michael Neumayer, Volker Paulsen, Kurt Roeckx, Vesa Ruokonen, Magnus Tjernstrom, Stefan Zehl.

10. References

[KEYWORDS] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[ABNF] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[IRC] Oikarinen, J. and D. Reed, "Internet Relay Chat Protocol", RFC 1459, May 1993.

[IRC-ARCH] Kalt, C., "Internet Relay Chat: Architecture", RFC 2810, April 2000.

[IRC-CLIENT] Kalt, C., "Internet Relay Chat: Client Protocol", RFC 2812, April 2000.

[IRC-CHAN] Kalt, C., "Internet Relay Chat: Channel Management", RFC 2811, April 2000.

[ZLIB] Deutsch, P. and J-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, May 1996.

[DEFLATE] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, May 1996.

[GZIP] Deutsch, P., "GZIP file format specification version 4.3", RFC 1952, May 1996.

[IDENT] St. Johns, M., "The Identification Protocol", RFC 1413, February 1993.

[TLS] Dierks, T. and C. Allen, "The TLS Protocol", RFC 2246, January 1999.

11. Author's Address

Christophe Kalt
99 Teaneck Rd, Apt #117
Ridgefield Park, NJ 07660
USA

EMail: kalt@stealth.net

12. Full Copyright Statement

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.