IRC
Internet Relay Chat

Finlandia, 1988,l'idea di IRC prende ispirazione dai primi software quali “MUT” e “Talk”. Da un idea di uno studente finlandese, questa innovazione pian piano si sviluppò in una rete IRC universitaria che permetteva agli studenti di tutte le università del paese di comunicare tramite i loro terminali Unix. Quando divenne disponibile la connessione Internet tra più stati la rete IRC Finlandese si espanse fino a raggiungere alcuni server Statunitensi dando vita poco tempo dopo alla prima rete IRC Internazionale tutt'oggi esistente: Eris Free Net (EfNet).

Ben presto, nel giro di poco tempo si ebbe un boom delle reti IRC Nazionali ed Internazionali, con una conseguente crescita esponenziale del numero di utenti nel mondo, tanto da far diventare IRC il sistema di Chat più diffuso.

In italia fece la prima comparsa intorno al 1995.

L'idea di IRC fu progettato come un sistema di chat "grezzo", dove non ci fosse alcun controllo sulle attività degli utenti. La mancanza di moderazione fu la causa principale della nascita e il conseguente sviluppo del fenomeno dell'IRCWar, che consiste nell'utilizzare tecniche più o meno lecite che vanno dall'attacco della macchina dell'utente fino al disturbo/attacco della stessa chatrom con lo scopo dell'impossersarsene.

L'indirizzo principale per collegarsi alla rete di solito è irc.xxx.xxx, quest'ultimo punta a rotazione su tutti i server principali per bilanciarne il carico a favore della stabilità e affidabilità della connessione.

INDICE:

1.SERVER

2.USO DI IRC
- 2.1 IDENTIFICAZIONE DELL'UTENTE
- 2.2 CANALI
- 2.3 REGOLE DI UNA ROM
- 2.4 INTERAZIONE TRA OPS E UTENTI
- 2.5 GLI IRCOP
- 2.6 INTERAZIONE TRA UTENTI
  - 2.6.1 CTCP
- 2.7 RACCOLTA DI COMANDI GENERICI DI UN CLIENT IRC
- 2.8 ALCUNI CLIENT IRC

3.EGGDROP
- 3.1INTRODUZIONE
- 3.2 CONFIGURAZIONE
  - 3.2.1 CONFIG ORIGINALE
  - 3.2.2 BREVE GUIDA AL CONFIG
- 3.3 USO
- 3.4 LINKARE + EGGDROP INSIEME

4.PSYBNC
- 4.1 INSTALLAZIONE
- 4.2 COMANDI

5.DA IPV4 A IPV6
- 5.1 CREAZIONE DI UN TUNNEL IPV6
- 5.2 COLLEGAMENTO DIRETTO AD UN SERVER IPV6

ALLEGATI:

- RFC 1459 UFFICIALE DELLA IETF
- TRATTO DI TRADUZIONE RFC 1459

1.SERVER IRC

Un server IRC è sostanzialmente un computer in grado di ricevere ed inviare in tempo reale messaggi di testo, servendosi del protocollo di trasmissione RFC1459 (il protocollo è uno standard in base al quale vengono trasmessi i dati in una rete di computer). Poiché il traffico di messaggi è notevole, sono impiegati computer multiprocessore con sistema operativo Unix; essi vengono gestiti da operatori (IRCop) attraverso un normale client (il programma utilizzato per collegarsi ad IRC) dotato però di particolari privilegi: si possono così gestire le connessioni al server (utenti collegati) e, se necessario, limitarle.

Un server IRC accetta tre differenti tipi di connessioni:

* client: normalmente utilizzati dagli utenti.
* server: per la formazione di reti IRC.
* service: speciali programmi in grado di eseguire statistiche o di svolgere altre funzioni.

Sebbene un solo server IRC sia in grado di gestire contemporaneamente i tre differenti tipi di connessione, è preferibile avere differenti server, ognuno ottimizzato per lo svolgimento di un compito specifico.

2.USO DI IRC

Sono preceduti dal carattere "/" (slash) per poter essere distinti dai messaggi veri e propri.

2.1 Identificazione dell'utente

nick is ~userid@host * realname
nick on #canale1 #canale2 #canale3 ..... #canalen
nick using nomeserver
nick utente registrato/non registrato (a seconda della versione del server)
nick End of /WHOIS list.

Dove:
- nick: è il sopranome dell'utente
- userid: è l'identita' dell'user impostato nel client irc
- realname: è il nome reale dell'utente
- on #canale: è i canali dove è presente l'utente

nota: se il canale dove è presente l'utente a cui è stato fatta la domanda "whois" è impostato nella modalità "secret",non sarà reso noto ad eccezione se è presente nello stesso canale anche l'utente che ha fatto la richiesta di "whois"
-using nomeserver: specifica il server a cui è connesso l'utente.
-stato: se il server ha la funzione di registrazione del nick, notifica se l'utente a cui è associato quel nick è suo perchè lo ha registrato, lo usa ma non è di sua la registrazione, oppure usa un nick non registrato.
-End of /Whois list: fine delle informazioni disponibili

2.2 Canali

I nomi dei canali di solito iniziano con il simbolo #
ESEMPIO: #GLM
e sharano insieme agli utenti a tutti i server irc della net, fa eccezione i canali con davanti il simbolo & che vengono creati solamente nel server locale.

Per entrare in un canale, o per crearne uno nuovo si usa il comando /join #nomecanale

Il primo che entra in una Rom nuova diventa subito proprietario prenderndo in automatico l'op che sta a significare "OPERATORE" del canale.
L'operatore verrà identificato dal nick preceduto dal simbolo della chiocciola:
@nick
L'operatore ha diritto di impostare le regole del canale e i permessi degli utenti.

2.3 Regole di una room

- Only ops set topic: solamente l'operatore può impostare il topic
- No external message: non possono essere visibili i messaggi provenienti dall'esterno del canale.
- Invite only: si può entrare nella stanza solamente tramite un invito.

/invite nick #canale
- Moderated: il canale è moderato, possono parlare solamente gli operatori e chi ha il

permesso di voice (flag +v)
- Key: si può entrare nel canale solamente tramite una chiave segreta:

/join #canale secretword
- Limit to: limite massimo di utenti presenti nel canale
- Private: il canale è impostato in modalità privato
- Secret: il canale è impostato in modalità segreta, ovvero alla richiesta di un "/whois" da un utente esterno al canale non sarà visualizzato il canale.
- Topic: in origine il topic era usato per descrivere il tema di discussione nella stanza, ma di solito viene usato in altri modi, che può variare dal comune messaggio di benveuto a qualsiasi altro argomento.

2.4 Interazione tra ops e utente

- Voice/Devoice: l'operatore da voce all'utente,in questo modo l'utente può parlare anche se il canale è moderato.
- Op/Deop: l'operatore da la chiocciola @ di controllo ad un altro utente del canale.
- Kick: l'operatore da un calcio all'untente cacciandolo dalla stanza.
- Ban: l'operatore aggiunge un utente nella lista BAN,ovvero una lista che non permette l'ingresso alla stanza alle persone indesiderate.

Nota: Il comando Ban aggiuge solo le info dell'utente nella lista degli indesiderati, ma non caccia l'utente, quindi la procedura giusta è il Ban+Kick, ovvero prima lo si aggiunge nella lista ban e poi lo si caccia.

2.5 Gli IRCOP

Ufficialmente la parola IRCOP sta a significare IRC Operator, ovvero operatori dell'intero network, ma spesso vengono interpretati in maniera errata come IRC-COP, ovvero polizia della rete IRC, solamente per i loro privilegi speciali.

Sono utenti con flag speciali e vengono nominati dall'IRC admin con lo scopo di monitorare la netnetique degli utenti.

Hanno varie caratteristiche come possibilità di opparsi da soli in qualsiasi canale.Una funzione molto importante che hanno è quella del Kline, ovvero hanno la possibilità di disconnettere un utente dal server per un periodo limitato/illimitato.

2.6 INTERAZIONE TRA UTENTI

Un utente può chattare o nel canale in pubblico, ovvero tutti leggeranno quello che si sta scrivendo, oppure in query, ovvero può rivolgersi in un modo privato direttamente ad un altro utente.

2.6.1 CTCP

CTCP significa Protocollo Client - Client (Client to Client Protocol).
Il traffico viene spedito ed interpretato direttamente tra i client IRC, senza che il server IRC vi faccia da tramite.

- DCC SEND, DIRECT CLIENT to CLIENTSEND, una connessione diretta al client allo scopo di sendare, fare un trasferimento di un file.
- DCC CHAT, DIRECT CLIENT to CLIENT CHAT: una connessione diretta al client allo scopo di chattare, senza passare per il server irc.

NOTA: La connessione dcc persisite anche una volta sconnessi dal server IRC in quanto è una connessione punto punto tra i due client.

2.7 I RACCOLTA DI COMANDI GENERICI DI UN CLIENT IRC

Come detto in precedenza ricordiamo che i comandi che il client invierà al server devono essere preceduti dal carattere "/" (slash) per poter essere distinti dai messaggi veri e propri.

/admin <server>

Elenca una serie di informazioni relative agli amministratori del server indicato.

/amsg <message>

Con questo comando è possibile inviare uno specifico messaggio a tutti i canali a cui si è collegati in quel momento.

/away [messaggio]

Per segnalare una momentanea assenza dalla discussione si puo' usare questo semplilce comando, che evita di uscire e ricollegarsi. E' consuigliato solo per brevi assenze ed un gesto di cortesia e correttezza verso gli utenti del canale.

/ban <nickname>

Consente la Chanop di bandire un utente da un canale.

/clear

Ripulisce la finestra, eseguendo uno scrollback della schermata corrente.

/dcc send <nickname> [filename]

(DCC=Direct Connection to remote Client).

Questo comando permette di stabilire una connessione diretta tra utenti bypassando completamente il server. Permette a due utenti di stabilire un collegamento remoto diretto, parlare ed inviarsi file. E' possibile anche inviare più file contemporaneamente usando la sintassi:

Esempio:

/dcc send <nickname> file1 file2 file3 ... fileN

/dcc get <nickname> [filename]

Permette ad un utente di ricevere i file da un altro utente

/dcc close

Causa la chiusura della sessione di trasferimento.

/dcc list

Elenca i trasferimetni in atto.

/dcc chat

Permette la conversazione diretta tra due utenti senza che i messaggi passino su IRC

/exit

Chiude il collegamento ed esce da irc.

/finger <nick/address>

Permette di eseguire il comando finger su di un utente collegato. Verranno mostrati una serie di dati come l'ultima data di connessione, l'IP Address e molti altri.

/disconnect <text>

Permette la disconnessione immediata dal Server. E' diverso (e più brusco) del comando / quit, che invia la richiesta al server e attende di essre disconnesso.

/fserve <nickname> [maxgets] [homedirectory] [welcomefile]

Permette di aprire una sessione fileserver con unaltro utente. In questo modo sarà possbile ad esempio condividere i dati presenti sul disco di un utente remoto utilizzando una serie di comandi del tutto simili a quelli utilizzati durante una sessione ftp.

Comandi in una sessione di tipo fserver:

cd [directory] - per cambiare la directory

dir [-b|k] [-#] [/w] - per listare il contenuto di una directory, supporta gli attributi indicati

ls [-b|k] [-#] - lista i file della directory corrente

get [filename] - invia un file specifico durante la connessione DCC

read [-numlines] [filename.txt] - permette di editare e leggere un file indicando anche il numero massimo di linee da visualizzare.

help - mostra i comandi a disposizione.

exit oppure bye - chiude la connessione

L'attributo maxgets indica il numero massimo di file inviati durante la connessione. Il welcome file è un file inviato all'utente appena la connessione viene attivata.

Ad esempio:

/fserve skywalker 5 \utente\saluti.txt

/help

Per chiedere aiuto e spiegazioni sulla sintassi dei comandi. Rappresenta sempre il modo migliore per imparare ad utilizzare in maniera approfondita i comandi di IRC in tutta la loro potenzialità.

/list

Permette di visualizzare tutti i canali pubblici aperti (attivi) sul server in quel momento, con l'argomento trattato ed il numero di utenti che vi partecipano. I canali segreti non vengono visualizzati, occorrerà conoscere a priori il loro nome. Per elenchi lunghi conviene usare set hold_mode on per poter esaminare una schermata alla volta oppure specificare di lista solo i canali aventi un numero minimo e un numero massimo di utenti utilizzando questa sintassi:

/list # (stringa) -min (numero minimo) -max (numero massimo)

Attivando l'opzione set hold mode si può far scorrere automaticamente la lista una schermata per volta:

/set hold_mode on

Inoltre si possono ricercare i canali contenenti una stringa particolare,che sara' rappresentata da una qualsiasi parola, la ricerca selezionera' solo i canali aventi questa stringa nel proprio topic.

Esempio:

/list -min 10 -max 50

Vengono listati i canali aventi un numero di utenti compresi tra i numeri 10 e 50.

/list #irc

Verranno indicati i canali contenenti la stringa irc nel proprio topic.

/log [on|off] <windowname>

Attiva la funzione di logging per una specifica finestra

/nick <alias>

Permette di modificare il nick, sostituendo alias con il nuovo nickname scelto

/join #canale

Per entrare in un canale, si usa questo comando: dove a canale va sostituito il nome del canale scelto. Se il canale digitato non esiste se ne crea uno nuovo all'istante.

/names #canale

Con questo comando è possibile ottenere una lista degli alias (nickname) dei partecipanti.

Uno degli alias, in genere, sarà preceduto dalla chiocciolina @ indicante il canale operator, ovvero colui che ha creato il canale. Solo lui può decidere chi accettare, se rendere segreto o privato il canale stesso.

/query <alias_a>

E' possibile aprire una conversazione privata con un utente utilizzando questo comando dove alias_a e' l'alias (o nickname) dell'interlocutore desiderato. Per rispondere ad una richiesta di query è necessario digitare /query alias_b, dove alias_b e' l'alias della persona che ha dato il precedente comando di query, e che ci invia la richiesta. Un metodo più semplice per stabilire una conversazione privata consiste nel fare doppio clic , sul nick interessato presente nella colonna destra della finestra di chat.

Per concludere la conversazione privata e' sufficiente digitare: /query

/invite <nickname|utente@hostname> #canale

Invita un utente ad entrare in un canale (eseguendo un join), ad esempio se è collegato su di un altro canale ma sullo stesso server.

/links

Mostra uin elenco dei server connessi alla rete.

/kick #nomecanale <nickname>

Permette a chi ha creato il canale, di espellere un utente collegato. E' possibile kickare un utente ed inviare un messaggio:

Esempio:

/kick #test nick Sei espulso!

/kill <nickname>

Permette a chi ha creato il canale, di killare un utente, in modo tale non solo da espellerlo dal canale (kick), ma forzando addirittura la disconnessione del suo modem dal collegamento con il provider.

/me [azione]

Specifica l'azione per un certo utente, ad esempio che cosa si sta facendo sul canale, oppure quale è l'opinione sull'argomento dibattuto.

Esempio:

/me Saluta tutti e vi augura buona notte.

/msg <nickname|canale> [testo]

Invia un messaggio privato ad un utente che non viene visualizzato da altri, evitando di aprire una finestra di conversazione privata.

/names canale

Fornisce una lista dei "soprannomi" utilizzati dagli utenti di ogni singolo canale.

/part oppure /leave canale

Serve per abbandonare un canale, ma non il server

/whowas <nickname>

Mostra una serie diinformazioni relative ad un utente che si è da poco scollegato al canale.

/whois #canale

Permette di apprendere informazioni sui partecipanti come l'e-mail dei partecipanti al canale:

/who #canale

Visualizza l'indirizzo e-mail dei partecipanti collegati sul canale indicato.

/mode #canale|nickname [[+|-]modechars [parameters]]

Questo è forse il comando più potente di IRC che consente di modificare i parametri del canale e degli utenti collegati. I comandi vengono impartiti utilizzando una sintassi complessa che permette di assegnare o togliere i valori (attributi), utilizzando il + o il -.

Sintassi del comando Mode Carattere Effetti sul canale

b impedisce di entrare nel canale a qualunque ad utenti il cui indirizzo sia "nick!user@host" corrisponda a

i permette di collegarsi al canale solo sotto invito del proprietario.

l rende il canale limitato ad un numero massimo di utenti.

m rende il canale moderato, cioè i messaggi inoltrati vengono filtrati. In pratica i messaggi giungono ai Chanop oppure agli utenti abilitati

n impedisce agli utenti del canale di inviare messaggi al di fuori di esso.

o permette ad utente identificato da uno specifico di diventare channel operator (proprietario del canale)

p rende il canale privato

s rende il canale segreto

t permette di impostare o cambiare il topic (argomento) del canale

k imposta delle chiavi segrete che gli utenti devono utilizzare per entrare del canale

2.8 ALCUNI CLIENT IRC

x-chat
http://www.xchat.org/

kv-irc
http://www.kvirc.net/

bitchx
http://www.bitchx.org/

epic
http://www.epicsol.org/

Zircon
http://catless.ncl.ac.uk/Programs/Zircon/

irssi
http://www.irssi.org/

QUirc
http://quirc.org/

3. EGGDROP

3.1 INTRODUZIONE

E' un programma scritto in C++, il suo scopo è quello di facilitare la gestione di una room di una IRC-CHAT, soprattutto quando il creatore della stanza non è online.
Un Eggdrop puo':
- Gestire gli utenti con specifiche regole di privilegi.
- Gestire i modi dei canali
- Gestire la lista BAN (la lista dei bannati, ovvero una lista di utenti che per un qualsiasi motivo sono stati cacciati)

NOTA: senza l'uso di un eggdrop, il ban in un canale IRC ha una breve durata (un tot di ore stabilito dagli admin del server IRC), mentre tramite una lista ban di un eggdrop, si puo' tenere un ban a time=0, ovvero a vita.

- Possibilità di chat privata tramite party-line (ovvero si crea un canale virtuale privato, riservato agli utenti addati, all'interno dell'eggdrop.)
- Possibilità di trasferimento dei file Utente/Eggdrop tramite una connessione CTCP DCC SEND:
- Possibilità di incrementare le funzioni di un Eggdrop tramite il caricamento delle TCL.

3.2 CONFIGURAZIONE

Segue il config originale e la spiegazione

3.2.1 CONFIG ORIGINALE

```
#! /path/to/executable/eggdrop
# ^- This should contain a fully qualified path to your Eggdrop executable.
#
# $Id: eggdrop.conf,v 1.7 2002/11/18 05:39:34 wcc Exp $
#
# This is a sample Eggdrop configuration file which includes all possible
# settings that can be used to configure your bot.
#
# More detailed descriptions of all these settings can be found in
# doc/settings/.


##### BASIC SETTINGS #####

# This setting defines the username the bot uses on IRC. This setting has
# no effect if an ident daemon is running on your bot's machine.
set username "lamest"

# This setting defines which contact person should be shown in .status,
# /msg help, and other places. You really should include this information.
set admin "Lamer <email: lamer@lamest.lame.org>"

# This setting is used only for info to share with others on your botnet.
# Set this to the IRC network your bot is connected to.
set network "I.didn't.edit.my.config.file.net"
```

```
# This setting defines the timezone is your bot in. It's used for internal
# routines as well as for logfile timestamping and scripting purposes.
# The timezone string specifies the name of the timezone and must be three
# or more alphabetic characters. For example, Central European Time(UTC+1)
# should be "CET".
set timezone "EST"

# The offset setting specifies the time value to be added to the local
# time to get Coordinated Universal Time (UTC aka GMT). The offset is
# positive if the local timezone is west of the Prime Meridian and
# negative if it is east. The value (in hours) must be between -23 and
# 23. For example, if the timezone is UTC+1, the offset is -1.
set offset "5"

# If you don't want to use the timezone setting for scripting purposes only,
# but instead everywhere possible, un-comment the following line.
#set env(TZ) "$timezone $offset"

# If you're using virtual hosting (your machine has more than 1 IP), you
# may want to specify the particular IP to bind to. You can specify either
# by hostname or by IP. You may also want to set the hostname here if
# Eggdrop has trouble detecting it when it starts up.
#set my-hostname "virtual.host.com"
#set my-ip "99.99.0.0"

# If you want to have your Eggdrop messages displayed in a language other
# than English, change this setting to match your preference. An alternative
# would be to set the environment variable EGG_LANG to that value.
#addlang "english"


##### LOG FILES #####

# Eggdrop is capable of logging various things, from channel chatter to
# commands people use on the bot and file transfers. Logfiles are normally
# kept for 24 hours. Afterwards, they will be renamed to "(logfile).yesterday".
# After 48 hours, they will be overwritten by the logfile of the next day.
#
# Events are logged by certain categories. This way, you can specify
# exactly what kind of events you want sent to various logfiles.
#
# The most common log file flags are:
#   m  private msgs/ctcps to the bot
#   k  kicks, bans, mode changes on the channel
#   j  joins, parts, netsplits on the channel
#   p  public chatter on the channel
#   s  server connects/disconnects/notices
#   b  information about bot linking and userfile sharing
#   c  commands people use (via msg or dcc)
#   x  file transfers and file-area commands
#      r  (if use-console-r enabled) EVERYTHING sent to the bot by the server
#   o  other: misc info, errors -- IMPORTANT STUFF
#   w  wallops: msgs between IRCops (be sure to set the bot +w in init-server)
#
# There are others, but you probably shouldn't log them, it'd be rather
# unethical. ;) There are also eight user-defined levels (1-8) which
# are used by Tcl scripts.
#
# Each logfile belongs to a certain channel. Events of type 'k', 'j', and 'p'
# are logged to whatever channel they happened on. Most other events are
```

```
# currently logged to every channel. You can make a logfile belong to all
# channels by assigning it to channel "*".

# This is the maximum number of logfiles allowed. This setting can be
# increased; however, don't decrease it.
set max-logs 5

# This is the maximum size of your logfiles. Set it to 0 to disable.
# This value is in kilobytes, so '550' would mean cycle logs when it
# reaches the size of 550 kilobytes. Note that this only works if you
# have keep-all-logs 0 (OFF).
set max-logsize 0

# This could be good if you have had problem with the logfile filling
# your quota/hard disk or if you log +p and publish it to the web and
# need more up-to-date info. Note that this setting might increase the
# CPU usage of your bot (on the other hand it will decrease your mem usage).
set quick-logs 0

# This creates a logfile named eggdrop.log containing private msgs/ctcps,
# commands, errors, and misc. info from any channel.
logfile mco * "logs/eggdrop.log"

# This creates a logfile named lamest.log containing joins, parts,
# netsplits, kicks, bans, mode changes, and public chat on the
# channel #lamest.
logfile jpk #lamest "logs/lamest.log"

# Use this feature to timestamp entries in the log file.
set log-time 1

# If you want to keep your logfiles forever, turn this setting on. All
# logfiles will get suffix ".[day, 2 digits][month, 3 letters][year, 4 digits]".
# Note that your quota/hard-disk might be filled by this, so check your logfiles
# often and download them.
set keep-all-logs 0

# If keep-all-logs is 1, this setting will define the suffix of the logfiles.
# The default will result in a suffix like "04May2000". "%Y%m%d" will produce
# the often used yyyymmdd format. Read the strftime manpages for more options.
# NOTE: On systems which don't support strftime, the default format will
# be used _always_.
set logfile-suffix ".%d%b%Y"

# You can specify when Eggdrop should switch logfiles and start fresh. You
# must use military time for this setting. 300 is the default, and describes
# 03:00 (AM).
set switch-logfiles-at 300

# "Writing user file..." and "Writing channel file..." messages won't be
# logged anymore if this option is enabled.
set quiet-save 0


##### CONSOLE #####

# This is the default console mode. It uses the same event flags as the log
# files do. The console channel is automatically set to your "primary" channel,
# which is set in the modules section of the config file. Masters can change
# their console channel and modes with the '.console' command.
```

```
set console "mkcobxs"


##### FILES AND DIRECTORIES #####

# Specify here the filename your userfile should be saved as.
set userfile "LamestBot.user"

# Specify here the filename Eggdrop will save its pid to. If no pidfile is
# specified, pid.(botnet-nick) will be used.
#set pidfile "pid.LamestBot"

# If you want your userfile to be sorted upon saving, enable this setting.
# This causes the bot to use bit more CPU when saving the usefile.
set sort-users 0

# Specify here where Eggdrop should look for help files. Don't modify this
# setting unless you know what you're doing!
set help-path "help/"

# Specify here where Eggdrop should look for text files. This is used for
# certain Tcl and DCC commands.
set text-path "text/"

# Set here a place to store temporary files.
set temp-path "/tmp"

# The MOTD (Message Of The day) is displayed when people dcc chat or telnet
# to the bot. Look at doc/text-substitutions.doc for options.
set motd "text/motd"

# This banner will be displayed on telnet connections. Look at
# doc/text-substitutions.doc for options.
set telnet-banner "text/banner"

# This specifies what permissions the user, channel, and notes files should
# be set to. The octal values are the same as for the chmod system command.
#
# To remind you:
#
#         u  g  o          u  g  o          u  g  o
#    0600  rw-------   0400  r--------   0200  -w-------    u - user
#    0660  rw-rw----   0440  r--r-----   0220  -w--w----    g - group
#    0666  rw-rw-rw-   0444  r--r--r--   0222  -w--w--w-    o - others
#
# Note that the default 0600 is the most secure one and should only be changed
# if you need your files for shell scripting or other external applications.
set userfile-perm 0600


##### BOTNET/DCC/TELNET #####

# Settings in this section should be unimportant for you until you deal
# with botnets (multiple Eggdrops connected together to maximize efficiency).
# You should read doc/BOTNET before modifying these settings.

# If you want to use a different nickname on the botnet than you use on
# IRC (i.e. if you're on an un-trusted botnet), un-comment the next line
# and set it to the nick you would like to use.
#set botnet-nick "LlamaBot"
```

```
# This opens a telnet port by which you and other bots can
# interact with the Eggdrop by telneting in.
#
# There are more options for the listen command in doc/tcl-commands.doc.
# Note, if you are running more than one bot on the same machine, you will
# want to space the telnet ports at LEAST 5 apart. 10 is even better.
#
# Valid ports are typically anything between 1025 and 65535 assuming the
# port is not already in use.
#

# If you would like the bot to listen for users and bots in separate ports,
# use the following format.
#
# listen 3333 bots
# listen 4444 users
#
# If you wish to use only one port, use this format:
listen 3333 all

# This setting defines whether or not people can boot users on the Eggdrop
# from other bots in your botnet. Valid settings are:
#   0 - allow *no* outside boots
#   1 - allow boots from sharebots
#   2 - allow any boots
set remote-boots 2

# This setting prohibits remote bots from telling your Eggdrop to unlink from
# share bots.
set share-unlinks 1

# This setting will drop telnet connections not matching a known host. It
# greatly improves protection from IRCops, but makes it impossible to add
# hosts on limbo (NOIRC) bots or have NEW as a valid login.
set protect-telnet 0

# This setting will make the bot ignore DCC chat requests which appear to
# have bogus information on the grounds that the user may have been trying
# to make the bot connect to somewhere that
 will get it into trouble, or
# that the user has a broken client (like mIRC tends to do), in which case
# the connect wouldn't work anyway.  It's suggested that you turn this on.
set dcc-sanitycheck 0

# This settings defines a time in seconds that the bot should wait before
# a dcc chat, telnet, or relay connection times out.
set ident-timeout 5

# Define here whether or not a +o user still needs the +p flag to dcc the bot.
set require-p 0

# If you want people allow to telnet in and type 'NEW' to become a new user,
# set this to 1. This is similar to the 'hello' msg command. The protect-telnet
# setting must be set to 0 to use this.
set open-telnets 0

# If you don't want Eggdrop to identify itself as an eggdrop on a telnet connection,
# set this setting to 1. Eggdrop will display 'Nickname' instead.
set stealth-telnets 0
```

```
# If you want Eggdrop to display a banner when telneting in, set this setting
# to 1. The telnet banner is set by 'set telnet-banner'.
set use-telnet-banner 0

# This settings defines a time in seconds that the bot should wait before
# a dcc chat, telnet, or relay connection times out.
set connect-timeout 15

# Specify here the number of lines to accept from a user on the partyline
# within 10 seconds before they are considered to be flooding and therefore
# get booted.
set dcc-flood-thr 3

# Define here how many telnet connection attempts in how many seconds from
# the same host constitute a flood. The correct format is Attempts:Seconds.
set telnet-flood 5:60

# If you want telnet-flood to apply even to +f users, set this setting to 1.
set paranoid-telnet-flood 1

# Set here the amount of seconds before giving up on hostname/address
# lookup (you might want to increase this if you are on a slow network).
set resolve-timeout 15


##### MORE ADVANCED SETTINGS #####

# Set this to your socks host if your Eggdrop sits behind a firewall.
# If you use a Sun "telnet passthru" firewall, use this setting:
#set firewall "!sun-barr.ebay:3666"

# If you have a NAT firewall (you box has an IP in one of the following
# ranges: 192.168.0.0-192.168.255.255, 172.16.0.0-172.31.255.255,
# 10.0.0.0-10.255.255.255 and your firewall transparently changes your
# address to a unique address for your box) or you have IP masquerading
# between you and the rest of the world, and /dcc chat,/ctcp chat or
# userfile sharing aren't working, enter your outside IP here. Do not
# enter anything for my-ip or my-hostname if you use this setting.
#set nat-ip "127.0.0.1"

# If you want all dcc file transfers to use a particular portrange either
# because you're behind a firewall, or for other security reasons, set it
# here.
#set reserved-portrange 2010:2020

# Set the time in minutes that temporary ignores should last.
set ignore-time 15

# Define here what Eggdrop considers 'hourly'. All calls to it, including such
# things as note notifying or userfile saving, are affected by this. For example:
#
# set hourly-updates 15
#
# The bot will save its userfile 15 minutes past every hour.
set hourly-updates 00

# Un-comment the next line and set the list of owners of the bot.
# You NEED to change this setting.
#set owner "MrLame, MrsLame"
```

```
# Who should a note be sent to when new users are learned?
set notify-newusers "$owner"

# Enter the flags that all new users should get by default. See '.help whois'
# on the partyline for a list of flags and their descriptions.
set default-flags "hp"

# Enter all user-defined fields that should be displayed in a '.whois'.
# This will only be shown if the user has one of these extra fields.
# You might prefer to comment this out and use the userinfo1.0.tcl script
# which provides commands for changing all of these.
set whois-fields "url birthday"

# Enable this setting if you want your Eggdrop to die upon receiving a SIGHUP
# kill signal. Otherwise, the Eggdrop will just save its userfile and rehash.
set die-on-sighup 0

# Enable this setting if you want your Eggdrop to die upon receiving a SIGTERM
# kill signal. Otherwise, the Eggdrop will just save its userfile and rehash.
set die-on-sigterm 1

# Comment these two lines if you wish to enable the .tcl and .set commands.
# If you select your owners wisely, you should be okay enabling these.
unbind dcc n tcl *dcc:tcl
unbind dcc n set *dcc:set

# If you enable this setting, only permanent owners (owner setting) will be
# able to use .tcl and .set. Moreover, if you want only let permanent owners
# use .dump, then set this to 2.
set must-be-owner 1

# Comment out this line to add the 'simul' partyline command (owners
# can manipulate other people on the party line). Please select owners
# wisely and use this command ethically!
unbind dcc n simul *dcc:simul

# Set here the maximum number of dcc connections you will allow. You can
# increase this later, but never decrease it.
set max-dcc 50

# Enable this setting if you want to enable the 'dccsimul' Tcl command.
set enable-simul 1

# Enable this setting if you want +d & +k users to use commands bound as -|-.
set allow-dk-cmds 1

# If your Eggdrop rejects bots that actually have already disconnected from
# another hub, but the disconnect information has not yet spread over the
# botnet due to lag, use this setting. The bot will wait dupwait-timeout
# seconds before it checks again and then finally reject the bot.
set dupwait-timeout 5

# You MUST remove this line for your bot to start. This has been added to
# prevent you from starting up a bot that is not fully configured. Bots
# that have not been fully configured may join the wrong IRC network, the
# wrong channels, or generally do things that you do not want. Please make
# sure that you have double-checked every setting. There's also a similar line
# lower down, just to make sure you're reading :)
die "Please make sure you edit your config file completely."
```

```
##### MODULES #####

# Below are various settings for the modules included with Eggdrop.
# PLEASE READ AND EDIT THEM CAREFULLY, even if you're an old hand at
# Eggdrop, things change.

# This path specifies the path were Eggdrop should look for its modules.
# If you run the bot from the compilation directory, you will want to set
# this to "". If you use 'make install' (like all good kiddies do ;), this
# is a fine default. Otherwise, use your head :)
set mod-path "modules/"


#### DNS MODULE ####

# This module provides asynchronous dns support. This will avoid long
# periods where the bot just hangs there, waiting for a hostname to
# resolve, which will often let it timeout on all other connections.
loadmodule dns

#### CHANNELS MODULE ####

# This module provides channel related support for the bot. Without it,
# you won't be able to make the bot join a channel or save channel
# specific userfile information.
loadmodule channels

# Enter here the filename where dynamic channel settings are stored.
set chanfile "LamestBot.chan"

# Set this setting to 1 if you want your bot to expire bans/exempts/invites set
# by other opped bots on the channel.
set force-expire 0

# Set this setting to 1 if you want your bot to share user greets with other
# bots on the channel if sharing user data.
set share-greet 0

# Set this setting to 1 if you want to allow users to store an info line.
set use-info 1

# The following settings are used as default values when you .+chan #chan or .tcl
# channel add #chan. Look in the section below for explanation of every option.

set global-flood-chan 10:60
set global-flood-deop 3:10
set global-flood-kick 3:10
set global-flood-join 5:60
set global-flood-ctcp 3:60
set global-flood-nick 5:60
set global-aop-delay 5:30
set global-idle-kick 0
set global-chanmode "nt"
set global-stopnethack-mode 0
set global-revenge-mode 1
set global-ban-time 120
set global-exempt-time 60
set global-invite-time 60

set global-chanset {
```

```
        -autoop         -autovoice
        -bitch          +cycle
        +dontkickops    +dynamicbans
        +dynamicexempts +dynamicinvites
        -enforcebans    +greet
        -inactive       -nodesynch
        -protectfriends +protectops
        -revenge        -revengebot
        -secret         -seen
        +shared         +statuslog
        +userbans       +userexempts
        +userinvites    -protecthalfops
        -autohalfop
}

# Add each static channel you want your bot to sit in using the following
# command. There are many different possible settings you can insert into
# this command, which are explained below.
#
#    channel add #lamest {
#      chanmode "+nt-likm"
#      idle-kick 0
#      stopnethack-mode 0
#      revenge-mode 1
#      ban-time 120
#      exempt-time 60
#      invite-time 60
#      aop-delay 5:30
#      need-op { putserv "PRIVMSG #lamest :op me cos i'm lame!" }
#      need-invite { putserv "PRIVMSG #lamest :let me in!" }
#      need-key { putserv "PRIVMSG #lamest :let me in!" }
#      need-unban { putserv "PRIVMSG #lamest :let me in!" }
#      need-limit { putserv "PRIVMSG #lamest :let me in!" }
#      flood-chan 10:60
#      flood-deop 3:10
#      flood-kick 3:10
#      flood-join 5:60
#      flood-ctcp 3:60
#      flood-nick 5:60
#    }
#
# chanmode +/-<modes>
#    This setting makes the bot enforce channel modes. It will always add
#    the +<modes> and remove the -<modes> modes.
# idle-kick 0
#    This setting will make the bot check every minute for idle
#    users. Set this to 0 to disable idle check.
#
# stopnethack-mode 0
#    This setting will make the bot de-op anyone who enters the channel
#    with serverops. There're seven different modes for this settings:
#       0 turn off,
#      1 isoptest (allow serverop if registered op)
#       2 wasoptest (allow serverop if op before split)
#       3 allow serverop if isop or wasop
#      4 allow serverop if isop and wasop.
#      5 If the channel is -bitch, see stopnethack-mode 3
#        If the channel is +bitch, see stopnethack-mode 1
#      6 If the channel is -bitch,
 see stopnethack-mode 2
```

```
#       If the channel is +bitch, see stopnethack-mode 4
#
# revenge-mode 1
#   This settings defines how the bot should punish bad users when
#   revenging. There are four possible settings:
#     0 Deop the user.
#     1 Deop the user and give them the +d flag for the channel.
#     2 Deop the user, give them the +d flag for the channel, and kick them.
#     3 Deop the user, give them the +d flag for the channel, kick, and ban them.
#
# ban-time 120
#   Set here how long temporary bans will last (in minutes). If you
#   set this setting to 0, the bot will never remove them.
#
# exempt-time 60
#   Set here how long temporary exempts will last (in minutes). If you
#   set this setting to 0, the bot will never remove them. The bot will
#   check the exempts every X minutes, but will not remove the exempt if
#   a ban is set on the channel that matches that exempt. Once the ban is
#   removed, then the exempt will be removed the next time the bot checks.
#   Please note that this is an IRCnet feature.
#
# invite-time 60
#   Set here how long temporary invites will last (in minutes). If you
#   set this setting to 0, the bot will never remove them. The bot will
#   check the invites every X minutes, but will not remove the invite if
#   a channel is set to +i. Once the channel is -i then the invite will be
#   removed the next time the bot checks. Please note that this is an IRCnet
#   feature.
#
# aop-delay (minimum:maximum)
# This is used for autoop, autohalfop, autovoice. If an op or voice joins a
# channel while another op or voice is pending, the bot will attempt to put
# both modes on one line.
#   aop-delay 0   No delay is used.
#   aop-delay X   An X second delay is used.
#   aop-delay X:Y A random delay between X and Y is used.
#
# need-op { putserv "PRIVMSG #lamest :op me cos i'm lame!" }
#    This setting will make the bot run the script enclosed in brackets
#    if it does not have ops. This must be shorter than 120 characters.
#    If you use scripts like getops.tcl or botnetop.tcl, you don't need
#    to set this setting.
#
# need-invite { putserv "PRIVMSG #lamest :let me in!" }
#    This setting will make the bot run the script enclosed in brackets
#    if it needs an invite to the channel. This must be shorter than 120
#    characters. If you use scripts like getops.tcl or botnetop.tcl, you
#    don't need to set this setting.
#
# need-key { putserv "PRIVMSG #lamest :let me in!" }
#    This setting will make the bot run the script enclosed in brackets
#    if it needs the key to the channel. This must be shorter than 120
#    characters. If you use scripts like getops.tcl or botnetop.tcl, you
#    don't need to set this setting
#
# need-unban { putserv "PRIVMSG #lamest :let me in!" }
#    This setting will make the bot run the script enclosed in brackets
#    if it needs to be unbanned on the channel. This must be shorter than
#    120 characters. If you use scripts like getops.tcl or botnetop.tcl,
```

```
#    you don't need to set this setting
#
# need-limit { putserv "PRIVMSG #lamest :let me in!" }
#    This setting will make the bot run the script enclosed in brackets
#    if it needs the limit to be raised on the channel. This must be
#    shorter than 120 characters. If you use scripts like getops.tcl or
#    botnetop.tcl, you don't need to set this setting
#
# flood-chan 10:60
#    Set here how many channel messages in how many seconds from one
#    host constitutes a flood. Setting this to 0 or 0:0 disables
#    flood protection for the channel.
#
# flood-deop 3:10
#    Set here how many deops in how many seconds from one host constitutes
#    a flood. Setting this to 0 or 0:0 disables deop flood protection for
#    the channel.
#
# flood-kick 3:10
#    Set here how many kicks in how many seconds from one host constitutes
#    a flood. Setting this to 0 or 0:0 disables kick flood protection for
#    the channel.
#
# flood-join 5:60
#    Set here how many joins in how many seconds from one host constitutes
#    a flood. Setting this to 0 or 0:0 disables join flood protection for
#    the channel.
#
# flood-ctcp 3:60
#    Set here how many channel ctcps in how many seconds from one host
#    constitutes a flood. Setting this to 0 or 0:0 disables ctcp flood
#    protection for the channel.
#
# flood-nick 5:60
#    Set here how many nick changes in how many seconds from one host
#    constitutes a flood. Setting this to 0 or 0:0 disables nick flood
#    protection for the channel.
#
# There are many different options for channels which you can
# define. They can be enabled or disabled using the channel set command by a
# plus or minus in front of them.
#
#   channel set #lamest +enforcebans +dynamicbans +userbans +dynamicexempts
#   channel set #lamest +userexempts +dynamicinvites +userinvites +protectops
#   channel set #lamest +protectfriends +statuslog +revenge -protecthalfops
#   channel set #lamest +revengebot +dontkickops +autovoice -autoop -autohalfop
#   channel set #lamest -bitch -secret -shared +greet +cycle
#
# A complete list of all available channel settings:
#
# enforcebans
#    When a ban is set, kick people who are on the channel and match
#    the ban?
#
# dynamicbans
#    Only activate bans on the channel when necessary? This keeps
#    the channel's ban list from getting excessively long. The bot
#    still remembers every ban, but it only activates a ban on the
#    channel when it sees someone join who matches that ban.
#
```

```
# userbans
#    Allow bans to be made by users directly? If turned off, the bot
#    will require all bans to be made through the bot's console.
#
# dynamicexempts
#    Only activate exempts on the channel when necessary? This keeps
#    the channel's exempt list from getting excessively long. The bot
#    still remembers every exempt, but it only activates a exempt on
#    the channel when it sees a ban set that matches the exempt. The
#    exempt remains active on the channel for as long as the ban is
#    still active.
#
# userexempts
#    Allow exempts to be made by users directly? If turned off, the
#    bot will require all exempts to be made through the bot's console.
#
# dynamicinvites
#    Only activate invites on the channel when necessary? This keeps
#    the channel's invite list from getting excessively long. The bot
#    still remembers every invite, but the invites are only activated
#    when the channel is set to invite only and a user joins after
#    requesting an invite. Once set, the invite remains until the
#    channel goes to -i.
#
# userinvites
#    Allow invites to be made by users directly? If turned off, the
#    bot will require all invites to be made through the bot's console.
#
# autoop
#    Op users with the +o flag as soon as they join the channel?
#    This is insecure and not recommended.
#
# autohalfop
#    Halfop users with the +l flag as soon as they join the channel?
#    This is insecure and not recommended.
#
# bitch
#    Only let users with +o) flag be opped on the channel?
#
# greet
#    Say a user's info line when they join the channel?
#
# protectops
#    Re-op a user with the +o flag if they get deopped?
#
# protecthalfops
#    Re-halfop a user with the +l flag if they get dehalfopped?
#
# protectfriends
#    Re-op a user with the +f flag if they get deopped?
#
# statuslog
#    Log the channel status line every 5 minutes? This shows the bot's
#    status on the channel (op, voice, etc.), The channel's modes, and
#    the number of +m/+o/+v/+n/+b/+e/+I users on the channel. A sample
#    status line follows:
#
#    [01:40] @#lamest (+istn) : [m/1 o/1 v/4 n/7 b/1 e/5 I/7]
#
# revenge
```

```
#    Remember people who deop/kick/ban the bot, valid ops, or friends
#    and punish them? Users with the +f flag are exempt from revenge.
#
# revengebot
#    This is similar to to the 'revenge' option, but it only triggers
#    if a bot gets deopped, kicked or banned.
#
# autovoice
#    Voice users with the +v flag when they join the channel?
#
# secret
#    Prevent this channel from being listed on the botnet?
#
# shared
#    Share channel-related user info for this channel?
#
# cycle
#    Cycle the channel when it has no ops?
#
# dontkickops
#    Do you want the bot not to be able to kick users who have the +o
#    flag, letting them kick-flood for instance to protect the channel
#    against clone attacks.
#
# inactive
#    This prevents the bot from joining the channel (or makes it leave
#    the channel if it is already there). It can be useful to make the
#    bot leave a channel without losing its settings, channel-specific
#    user flags, channel bans, and without affecting sharing.
#
# seen
#    Respond to seen requests in the channel?  The seen module must be
#    loaded for this to work.
#
# nodesynch
#    Allow non-ops to perform channel modes? This can stop the bot from
#    fighting with services such as ChanServ, or from kicking IRCops when
#    setting channel modes without having ops.
#
# Here is a shorter example:
#
#   channel add #botcentral {
#     chanmode "+mntisl 1"
#     idle-kick 1
#   }
#   channel set #botcentral +bitch +enforcebans -greet +revenge


#### SERVER MODULE ####

# This module provides the core server support. You have to load this
# if you want your bot to come on IRC. Not loading this is equivalent
# to the old NO_IRC define.
loadmodule server

# What is your network?
#   0 = EFnet (non +e/+I hybrid)
#   1 = IRCnet
#   2 = Undernet
#   3 = DALnet
```

```
#   4 = EFnet +e/+I hybrid
#   5 = Others
set net-type 0

# Set the nick the bot uses on IRC, and on the botnet
# unless you specify a separate botnet-nick, here.

set nick "Lamestbot"

# Set the alternative nick which the bot uses on IRC if the nick specified
# by 'set nick' is unavailable. All '?' characters will be replaced by random
# numbers.

set altnick "Llamab?t"

# Set what should be displayed in the real-name field for the
 bot on IRC.
set realname "/msg LamestBot hello"

# Set here a script to run (if any) when first connecting to a server.
# This is limited to 120 characters.
set init-server { putserv "MODE $botnick +i-ws" }

# Set the default port which should be used if none is specified with
# '.jump' or in 'set servers'.
set default-port 6667

# This is the bot's server list. The bot will start at the first server listed,
# and cycle through them whenever it gets disconnected. You need to change these
# servers to YOUR network's servers.
#
# The format is:
#   server[:port[:password]]
#
# Both the port and password fields are optional; however, if you want to set a
# password you must also set a port. If a port isn't specified it will default to
# your default-port setting.
set servers {
  you.need.to.change.this:6667
  another.example.com:7000
}

# This setting makes the bot try to get his original nickname back if its
# primary nickname is already in use.
set keep-nick 1

# Set this to 1 if you don't want your the bot to strip a leading '~'
# on user@hosts.
set strict-host 0

# This setting makes the bot squelch the error message when rejecting a DCC
# CHAT or SEND. Normally it tells the DCC user that the CHAT or SEND has
# been rejected because they don't have access.
# Please note, sometimes IRC server operators detect bots that way.
set quiet-reject 1

# If you want your bot to answer lower case ctcp requests (non rfc-
# compliant), set this setting to 1. mIRC will do this, most other
# clients will not.
set lowercase-ctcp 0
```

```
# Set how many ctcps should be answered at once.
set answer-ctcp 3

# Set here how many msgs in how many seconds from one host constitutes
# a flood. If you set this to 0:0, msg flood protection will be disabled.
set flood-msg 5:60

# Set here how many ctcps in how many seconds from one host constitutes
# a flood. If you set this to 0:0, ctcp flood protection will be disabled.
set flood-ctcp 3:60

# This settings makes the bot cycle forever through the server list until
# it successfully connects to one.
set never-give-up 1

# Often, IRC servers call themselves a name other than their actual
# hostname. If you want Eggdrop to replace your entry in the config
# file with this name, set this to 1. If you set this setting to 0,
# Eggdrop will keep a separate list of what IRC servers call themselves.
set strict-servernames 0

# This setting defines how long Eggdrop should wait before moving from one
# server to another on disconnect. If you set 0 here, Eggdrop will not wait
# at all and will connect instantly. Setting this too low could result in
# your bot being K:Lined.
set server-cycle-wait 60

# Set here how long Eggdrop should wait for a response when connecting to a
# server before giving up and moving on to next server.
set server-timeout 60

# If the number of servers on the net gets below this number, the bot
# will jump to a new server (it will assume it's on the losing end of
# a netsplit). Set this to 0 to turn off. If your bot is running on
# any major IRC network, this should probably be turned off.
set servlimit 0

# Set this to 1 if Eggdrop should check for stoned servers? (where the
# server connection has died, but Eggdrop hasn't been notified yet).
set check-stoned 1

# This setting allows you the logging of all information received from the
# server via console mode 'r'.
# NOTE: This is a large security hole, allowing people to see user passwords.
# This is now restricted to +n users only. Please choose your owners with
# care.
set use-console-r 0

# This setting allows you the logging of all information sent to the server
# via console/log mode 'v'. It also allows logging of botnet traffic via
# console/log mode 't'.
# NOTE: This is a large security hole, allowing people to see passwords.
# This is now restricted to +n users only. Please choose your owners with
# care.
set debug-output 0

# If you want your bot to exit the server if it receives an ERROR message,
# set this to 1.
set serverror-quit 1
```

```
# Set here the maximum number of lines to queue to the server. If you're
# going to dump large chunks of text to people over IRC, you will probably
# want to raise this. 300 is fine for most people though.
set max-queue-msg 300

# If you want Eggdrop to trigger binds for ignored users, set this to 1.
set trigger-on-ignore 0

# Allow identical messages in the mode queue?
set double-mode 0

# Allow identical messages in the server queue?
set double-server 0

# Allow identical messages in the help queue?
set double-help 0

# This optimizes the kick queue. It also traces nick changes and parts in
# the channel and changes the kick queue accordingly. There are three
# different options for this setting:
#   0 = Turn it off.
#   1 = Optimize the kick queue by summarizing kicks.
#   2 = Trace nick changes and parts on the channel and change the queue
#       accordingly. For example, bot will not try to kick users who have
#       already parted the channel.
# ATTENTION: Setting 2 is very CPU intensive.
set optimize-kicks 1

# If your network supports more recipients per command then 1, you can
# change this behavior here. Set this to the number of recipients per
# command, or set this to 0 for unlimited.
set stack-limit 4

### SERVER MODULE - OTHER NETWORKS (net-type 5) ###

# This settings defines how umode +r is understood by Eggdrop. Some
# networks use +r to indicate a restricted connection. If this is your
# case, and you want your bot to leave restricted servers and jump to
# the next server on its list, then set it to 1.
#set check-mode-r 1

# This setting allows you to specify the maximum nick-length supported by
# your network. The default setting is 9. The maximum supported length by
# Eggdrop is 32.
#set nick-len 9

#### CTCP MODULE ####

# This module provides the normal ctcp replies that you'd expect.
# Without it loaded, CTCP CHAT will not work. The server module
# is required for this module to function.
loadmodule ctcp

# Set here how the ctcp module should answer ctcps. There are 3 possible
# operating modes:
#   0: Normal behavior is used.
#   1: The bot ignores all ctcps, except for CHAT and PING requests
#      by users with the +o flag.
#   2: Normal behavior is used, however the bot will not answer more
#      than X ctcps in Y seconds (defined by 'set flood-ctcp').
```

```
set ctcp-mode 0

# There are also several variables to help make your bot less noticeable.
# They are: ctcp-version, ctcp-finger, and ctcp-userinfo. You can use set to set
# them to values you'd like.


#### IRC MODULE ####

# This module provides basic IRC support for your bot. You have to
# load this if you want your bot to come on IRC. The server and channels
# modules must be loaded for this module to function.
loadmodule irc

# Set this to 1 if you want to bounce all server bans.
set bounce-bans 1

# Set this to 1 if you want to bounce all server modes.
set bounce-modes 0

# Set here the maximum number of bans you want the bot to set on a channel.
# Eggdrop will not place any more bans if this limit is reached. Undernet
# and IRCnet currently allow 30 bans, EFnet allows 20, and DALnet allows 100.
set max-bans 20

# There is a global limit for +b/+e/+I modes. This limit is currently set to
# 30 on IRCu 2.10 servers.
set max-modes 30

# Set this to 1 if you want the bot to kick for control character/ctcp
# avalanches to a channel. Remember that if it does, it won't ban them.
# This can start kick floods.
set kick-fun 0

# Set this to 1 if you want the bot to ban for control character/ctcp
# avalanches to a channel. This can prevent kick floods, but it also can
# fill the banlist.
set ban-fun 0

# If you want people to be able to add themselves to the bot's userlist
# with the default userflags (defined above in the config file) via the
# 'hello' msg command, set this to 1.
set learn-users 0

# Set here the time (in seconds) to wait for someone to return from
# a netsplit (i.e. wasop will expire afterwards). Set this to 1500
# on IRCnet since its nick delay stops after 30 minutes.
set wait-split 600

# Set here the time (in seconds) that someone must have been off-channel
# before re-displaying their info line.
set wait-info 180

# Set this to the maximum number of bytes to send in the arguments
# of modes sent to the server. Most servers default this to 200.
set mode-buf-length 200

# Many IRCops find bots by seeing if they reply to 'hello' in a msg.
# You can change this to another word by un-commenting the following
# two lines and changing "myword" to the word wish to use instead of
```

```
# 'hello'. It must be a single word.
#unbind msg - hello *msg:hello
#bind msg - myword *msg:hello

# Many takeover attempts occur due to lame users blindly /msg ident'ing to
# the bot and attempting to guess passwords. We now unbind this command by
# default to discourage them. You can enable this command by un-commenting
# the following two lines.
unbind msg - ident *msg:ident
unbind msg - addhost *msg:addhost

# If you are so lame you want the bot to display peoples info lines, even
# when you are too lazy to add their chanrecs to a channel, set this to 1.
# *NOTE* This means *every* user with an info line will have their info
# line displayed on EVERY channel they join (provided they have been gone
# longer than wait-info).
set no-chanrec-info 0

### IRC MODULE - IRCnet SPECIFIC FEATURES (net-type 1) ###

# Attention: Use these settings *only* if you set 'net-type' to 1!

# Set this to 1 if you want to bounce all server exemptions (+e modes).
set bounce-exempts 0

# Set this to 1 if you want to bounce all server invitations (+I modes).
set bounce-invites 0

# Set here the maximum number of exempts you want Eggdrop to set
# on a channel. Eggdrop will not place any more exempts if this
# limit is reached.
set max-exempts 20

# Set here the maximum number of invites you want Eggdrop to set
# on a channel. Eggdrop will not place
 any more invites if this
# limit is reached.
set max-invites 20

# The following settings should be left commented unless the default values
# are being overridden. By default, exempts and invites are on for IRCnet,
# but off for all other large networks. This behavior can be modified with
# the following 2 flags. If your network doesn't support +e/+I modes then you
# will be unable to use these features.
#
# Do you want to enable exempts?
#set use-exempts 0

# Do you want to use invites?
#set use-invites 0

# At the moment, the current IRCnet IRCd version (2.10) doesn't support the mixing
# of b,o and v modes with e and I modes. This might be changed in the future, so
# use 1 at the moment for this setting.
set prevent-mixing 1

### IRC MODULE - OTHER NETWORKS (net-type 5) ###

# Attention: Use these settings *only* if you set 'net-type' to 5!
```

```
# If your network supports more users per kick command then 1, you can
# change this behavior here. Set this to the number of users to kick at
# once, or set this to 0 for all at once.
#set kick-method 1

# Some networks allow you to stack lots of channel modes into one line.
# They're all guaranteed to support at least 3, so that's the default.
# If you know your network supports more, you may want to adjust this.
# This setting is limited to 6, although if you want to use a higher value,
# you can modify this by changing the value of MODES_PER_LINE_MAX in
# src/chan.h and recompiling the bot.
#set modes-per-line 3

# Some networks don't include the +l limit and +k or -k key modes
# in the modes-per-line (see above) limitation. Set include-lk to 0 for
# these networks.
#set include-lk 1

# Set this to 1 if your network uses IRCu2.10.01 specific /who requests.
# Eggdrop can, therefore, ask only for exactly what's needed.
#set use-354 0

# If your network doesn't use rfc 1459 compliant string matching routines,
# set this to 0.
#set rfc-compliant 1


#### TRANSFER MODULE ####

# The transfer module provides dcc send/get support and userfile transfer
# support for userfile sharing. Un-comment the next line to load it if you
# need this functionality.
#loadmodule transfer

# Set here the maximum number of simultaneous downloads to allow for
# each user.
set max-dloads 3

# Set here the block size for dcc transfers. ircII uses 512 bytes,
# but admits that may be too small. 1024 is standard these days.
# Set this to 0 to use turbo-dcc (recommended).
set dcc-block 1024

# Enable this setting if you want to copy files into the /tmp directory
# before sending them. This is useful on most systems for file stability,
# but if your directories are NFS mounted, it's a pain, and you'll want
# to set this to 0. If you are low on disk space, you may also want to
# set this to 0.
set copy-to-tmp 1

# Set here the time (in seconds) to wait before an inactive transfer
# times out.
set xfer-timeout 30


#### SHARE MODULE ####

# This module provides userfile sharing support between two directly
# linked bots. The transfer and channels modules are required for this
# module to correctly function. Un-comment the following line to load
```

```
# the share module.
#loadmodule share

# Settings in this section must be un-commented before setting.

# When two bots get disconnected, this setting allows them to create a
# resync buffer which saves all changes done to the userfile during
# the disconnect. When they reconnect, they will not have to transfer
# the complete user file, but, instead, just send the resync buffer.
#
# NOTE: This has been known to cause loss of channel flags and other
# problems. Using this setting is not recommended.
#set allow-resync 0

# This setting specifies how long to hold another bots resync data
# before flushing it.
#set resync-time 900

# When sharing user lists, DON'T ACCEPT global flag changes from other bots?
# NOTE: The bot will still send changes made on the bot, it just won't accept
# any global flag changes from other bots.
#set private-global 0

# When sharing user lists, if private-global isn't set, which global flag
# changes from other bots should be ignored?
#set private-globals "mnot"

# When sharing user lists, don't accept ANY userfile changes from other
# bots? Paranoid people should use this feature on their hub bot. This
# will force all userlist changes to be made via the hub.
#set private-user 0

# This setting makes the bot discard its own bot records in favor of
# the ones sent by the hub.
# NOTE: No passwords or botflags are shared, only ports and
# address are added to sharing procedure. This only works with hubs that
# are v1.5.1 or higher.
#set override-bots 0


#### COMPRESS MODULE ####

# This module provides provides support for file compression. This allows the
# bot to transfer compressed user files and therefore save a significant amount
# of bandwidth. The share module must be loaded to load this module. Un-comment
# the following line to the compress module.
#loadmodule compress

# Allow compressed sending of user files? The user files are
# compressed with the compression level defined in `compress-level'.
set share-compressed 1

# This is the default compression level used.
#set compress-level 9


#### FILESYSTEM MODULE ####

# This module provides an area within the bot where users can store
# files. With this module, the bot is usable as a file server. The
```

```
# transfer module is required for this module to function. Un-comment
# the following line to load the filesys module.
#loadmodule filesys

# Set here the 'root' directory for the file system.
set files-path "/home/mydir/filesys"

# If you want to allow uploads, set this to the directory uploads
# should be put into. Set this to "" if you don't want people to
# upload files to your bot.
set incoming-path "/home/mydir/filesys/incoming"

# If you don't want to have a central incoming directory, but instead
# want uploads to go to the current directory that a user is in, set
# this setting to 1.
set upload-to-pwd 0

# Eggdrop creates a '.filedb' file in each subdirectory of your file area
# to keep track of its own file system information. If you can't do that (for
# example, if the dcc path isn't owned by you, or you just don't want it to do
# that) specify a path here where you'd like all of the database files to be
# stored instead.
set filedb-path ""

# Set here the maximum number of people that can be in the file area at once.
# Setting this to 0 makes it effectively infinite.
set max-file-users 20

# Set here the maximum allowable file size that will be received (in kb).
# Setting this to 0 makes it effectively infinite.
set max-filesize 1024


#### NOTES MODULE ####

# This module provides support for storing of notes for users from each
# other. Note sending between currently online users is supported in the
# core, this is only for storing the notes for later retrieval.
loadmodule notes

# Set here the filename where private notes between users are stored.
set notefile "LamestBot.notes"

# Set here the maximum number of notes to allow to be stored for
# each user (to prevent flooding).
set max-notes 50

# Set here how long (in days) to store notes before expiring them.
set note-life 60

# Set this to 1 if you want to allow users to specify a forwarding
# address for forwarding notes to another account on another bot.
set allow-fwd 0

# Set this to 1 if you want the bot to let people know hourly if they
# have any notes.
set notify-users 1

# Set this to 1 if you want the bot to let people know on join if they
# have any notes.
```

```
set notify-onjoin 1

# Comment out this next line. Otherwise, your bot won't start.
die "You didn't edit your config file completely like you were told, did you?"


#### CONSOLE MODULE ####

# This module provides storage of console settings when you exit the
# bot or type .store on the partyline.
loadmodule console

# Save users console settings automatically? Otherwise, they have
# to use the .store command.
set console-autosave 1

# If a user doesn't have any console settings saved, which channel
# do you want them automatically put on?
set force-channel 0

# Enable this setting if a user's global info line should be displayed
# when they join a botnet channel.
set info-party 0


#### WOOBIE MODULE ####

# This is for demonstrative purposes only. If you are looking for starting
# point in writing modules, woobie is the right thing.
#loadmodule woobie


#### SEEN MODULE ####

# This module provides very basic seen commands via msg, on channel or via dcc.
# This module works only for users in the bot's userlist. If you are looking for
# a better and more advanced seen module, try the gseen module by G'Quann. You
# can find it at http://www.visions-of-fantasy.de/gseen.mod/.
#loadmodule seen


#### BLOWFISH MODULE ####

# IF YOU DON'T READ THIS YOU MAY RENDER YOUR USERFILE USELESS LATER
# Eggdrop encrypts its userfile, so users can have secure passwords.
# Please note that when you change your encryption method later (i.e.
# using other modules like a md5 module), you can't use your current
# userfile anymore. Eggdrop will not start without an encryption module.
#loadmodule blowfish


#### ASSOC MODULE ####

# This module provides assoc support, i.e. naming channels on the botnet.
# You can load it by un-commenting the following line.
#loadmodule assoc


#### WIRE MODULE ####
```

```
# This module provides all the standard .wire commands via dcc. It is an
# encrypted partyline communication tool, compatible with wire.tcl. An
# encryption module must be loaded to use this module. Un-comment the
# following line to load the wire module.
#loadmodule wire

#### UPTIME MODULE ####

# This module reports uptime statistics to http://uptime.eggheads.org.
# Go look and see what your uptime is! It takes about 9 hours to show up,
# so if your bot isn't listed, try again later. The server module must be
# loaded for this module to function.
#
# Information sent to the server
 includes the bot's uptime, botnet-nick,
# server, version, and IP address. This information is stored in a temporary
# logfile for debugging purposes only. The only publicly available information
# will be the bot's botnet-nick, version and uptime. If you do not wish for this
# information to be sent, comment out the following line.
loadmodule uptime

##### SCRIPTS #####

# This is a good place to load scripts to use with your bot.

# This line loads script.tcl from the scripts directory inside your Eggdrop's
# directory. All scripts should be put there, although you can place them where
# you like as long as you can supply a fully qualified path to them.
#
# source scripts/script.tcl

source scripts/alltools.tcl
source scripts/action.fix.tcl

# Use this script for Tcl and Eggdrop downwards compatibility.
# NOTE: This can also cause problems with some newer scripts.
source scripts/compat.tcl

# This script provides many useful informational functions, like setting
# users' URLs, e-mail address, ICQ numbers, etc. You can modify it to add
# extra entries.
source scripts/userinfo.tcl
loadhelp userinfo.help
```

3.2.2 BREVE GUIDA AL CONFIG

set username (nome utente): se la propria shell manda l'idendt (molte lo fanno), bisogna settarlo col login del proprio account.

set my-hostname e set my-ip: bisognerà settarli se si vuole che il proprio bot usi un vhost (i vhost sono reperibili nelle info della shell, con un opportuno comando, che può cambiare da shell a shell si ottiene l'elenco dei vhost disponibili).
Il settaggio my-hostname riguarda il vhost, mentre my-ip è l' indirizzo IP abbinato al vhost, Non c'è bisogno di settarli entrambi, ma è raccomandabile in quanto può essere utile se la shell ha problemi col DNS.

logfile: tenere i logs è un buona idea. Di solito si può aver un log per ogni bot e un log per ciascun canale.

Per il bot: si aggiunga la linea: logfile mcobxs * "botnick.log" al file di configurazione.
Per i canali: si aggiunga: logfile jkp #donkeys "#donkeys.log", logfile jkp #horses "#horses.log", ecc.
Se si vuol mettere i file log nella propria directory, bisognerà specificala nel nome del log (x es. logfile jkp #donkeys "logs/#donkeys.log" per scrivere il file log nella directory /log).

set sort-users: di default le entrate dei fileusers sono ordinate per ciascun utente nell'ordine in cui è stato aggiunto, dal primo all'ultimo. Settandolo su 1 si baserà l'ordine della lista utenti sulle flag degli utenti. Entrambi i metodi sono utili - si consiglia di lasciare il settaggio 0 per cominciare.

listen 3333 all: si vorrà chiaramente modificarlo. Si scelga una porta tra 4000 e 65536. E' possibile disabilitarlo togliendo il commento(#), ma questo impedirà qualunque telnet al bot (non si sarà pertanto in grado di usare il bot come hub, non potrai fare un telnet al bot, e il bot non risponderà alle richieste /ctcp botnick CHAT). Si può settare invece, commentanto questa opzione una porta diversa per i bot e una per gli utenti.

set protect-telnet: settarlo su 1 è vivamente consigliato per ragioni di sicurezza.

set require-p: questo è un aspetto molto usato che permette di dare accesso alla party line ad un utente specifico. Si consiglia di settare 1.

set stealth-telnets: quando si fa un telnet al bot, solitamente mostrerà il proprio nickname e le informazioni sulla versione. Probabilmente non si vuole che la gente veda tali informazioni se fanno un portscan sulla shell del bot. Settando 1 si eviterà che il bot mostri il suo nickname e la proria versione quando qualcuno fa un telnet.

set dcc-flood-thr:questo settaggio determina il numero di linee per secondo che si possono spedire alla party line prima di esserne espulso. Può essere una scocciatura se si intende i inviare molte righe alla party line, di conseguenza è possibile aumentare il numero con qualcosa tipo 5 o 10.

set hourly-updates: è una buona idea cambiare il settaggio di default che è 00, dal momento che molti degli altri bot stanno usando 00 mettendo sotto forte stress la shell nello stesso momento. Si scelga qualcosa che non sia un multiplo di 10 (x es 03, 37 o 56 sono buoni settaggi).

set notify-newusers: lo si setti settalo con il nick che si avrà avrai sul bot. Questo settaggio non viene usato se non si è attivata learn-users.

set owner: si può mettere un sola persona in questa lista-se stessi. Ovviamente si imposti il nick che si avrà sul bot. NON lasciare l'impostazione di default "MrLame, MrsLame".

set default-flags: queste sono le flags assegnate automaticamente ad un utente quando entra nel bot (se learn-users è attivato) o quando sono addati usando .adduser.
Se non si vuole che agli utenti venga data alcuna flag inizialmente, si imposti "" oppure "-"

set remote-boots: l'impostazione di default è 2 ma può infastidire essere cacciati dalla party line (che è come essere kickati da un canale IRC). E' meglio settare con 0 oppure 1.

unbind dcc n tcl *dcc: tcl e unbind dcc n set *dcc:set: queste righe riguardano i comandi .tcl e .set. E' una buona idea lasciare queste righe da parte, in quanto i comandi .tcl e .set possono mettere a rischio la sicurezza dal momento che permettono l'accesso al tuo account di shell attraverso il bot.
Questi comandi sono molto utili solo se programmi con script per la scrittura in Tcl.

set must-be-owner: se hai i comandi .tcl e .set abilitati, si deve chiaramente impostarlo con 1. Nell'1.3.26 e precedenti, è meglio impostarlo con 2 per maggior sicurezza.

set chanfile: il chanfile permette di tenere sotto controllo i canali "dinamici" cosicchè il bot può rientrare nel canale una volta riavviato. I canali dinamici sono quelli in cui si può far entrare il bot col comando .+chan -essi non sono definiti nel file config.
Il chanfile è comodo se si rimuove/agginge frequentemente canali dal bot, ma può essere fastidioso se si vuol aggiungere/rimuovere i canali usando il file di configurazione dal momento che i settaggi nel chanfile sovrascrivono quelli di configurazione.
Si può scegliere di non usare il chanfile scrivendo ""

channel add: si deve usare per aggiungere canali al bot. Vi sono molte opzioni per questo comando. I canali sono aggiunti nel seguente formato:
channel add #donkeys {
options
}
channel set #donkeys +option -option
channel add #horses {
options
}
channel set #horses +option -option
Tutte le opzioni e le impostazioni sono mostrate negli esempi nel file di configurazione.
Ci si accerti di aver rimosso gli esempi per #lamest e #botcentral.

set nick: serve per specificare il nome del bot.Si consiglia di non usare i caretteri [] {}\ nel nome del bot, dal momento che possono causare problemi con alcuni script Tcl, ma se vuole usarli, bisogna far precedere ognuno di tali caratteri da un backslash, per es se si vuole che il bot si chiami [NiceBot], usa set nick "\[NiceBot\]".

set altnick: se si vuol usare i caratteri [ ] { } \ nel nick sostitutivo del bot, si segua la regola del backslash descritta prima.

set servers: si può specificare più server in questa lista, nel caso il bot non possa connettersi al primo server. Il formato della lista è mostrato di seguito:
set servers {
irc.chitchat.com:6667
irc.talkworld.com:6665
irc.nice.net:6667
}
Dove possibile, sarà opportuno usare una porta diversa dalla 6667 (ci si connetta al server e si digiti /motd per avere una lista delle porte utilizzabili).
E' fondamentale usufruire di server che permettano bot (alcune shell hanno delle regole per far rispettare questo), ma a meno che il contratto con la shell non preveda il contrario, si potranno usare server che non permettono i bot come appoggio in caso il network IRC abbia pochi server per bot, in modo che il bot possa connettersi (ma ovviamente si pongano server che lo permettono in cima alla lista).

set use-ison: si lasci il settaggio a 1, se si setta con 0 si farà usare al bot il comando 'trace' e ciò potrebbe comportare una K-line (ban) da parte del server.

set strict-host: se è settato a 0, l'Eggdrop ignorerà la tilde negli ident che iniziano con ~. Settare 1 è un pochino più sicuro, ma può essere difficoltoso per la gestione della hostmask dell'utente. Se non si è sicuri di come settare, è meglio lasciare 0.

set server-cycle-wait: per default l'Eggdrop attende 60 secondi per ogni tentativo di connessione al sever. E' un po' troppo tempo, ma è neccessario per evitare di 'essere respinti' dal server Undernet (se un server riceve troppe connessioni da un particolare host in un breve periodo di tempo, bloccherà tutte le connessioni da quell'host finchè non ci sai stata una pausa nei tentativi di connessione). Se si usa Undernet e si condivide una shell con molti altri utenti di bot, si lasci questa opzione su 60, altrimenti è una buona idea scendere a qualcosa come 20.

set trigger-on-ignore: impostando 1 si diminuisce la capacità dell'Eggdrop di ignorare. Si consiglia di lasciare 0.

set bounce-bans: impostato a 1 fa rimuovere al bot ogni ban impostato dal server.

set bounce-modes: impostato a 1 fa rimuovere al bot ogni mode impostato dal server.

set learn-users: questa è un'impostazione importante che determina quali utenti saranno addati all'Eggdrop. Se si imposta 1, la gente si potrà addare al bot scrivendo 'hello' (gli utenti saranno addati con la flag impostata nel default-flags). Se si imposta 0, gli utenti non potranno addarsi - dovranno addarli un master o un owner usando il comando . adduser

unbind msg - hello *msg:hello e bind msg - myword *msg:hello: permettono di cambiare il comando 'hello' con qualcosa di diverso. Cambia myword con la parola chesi vuol sostituire al comando 'hello'. Se si ha learn-users impostato a 1, questo comando è usato solo quando ci presenta al bot.

unbind msg - ident *msg: ident e unbind msg - addhost *msg:addhost: queste righe liberano i comandi ident e addhost.
Quasi certamente vorrai di nuovo si vorrà riabilitare il comando ident, sia commentando la riga unbind
(ce invece lascia il comando /msg ident di default) sia legandola ad un altro comando bind. E' una buona idea cambiare il comando per extra sicurezza. Per legarlo ad un altro comando, diciamo 'horse', si deve aggiungere la riga bind msg - horse *msg:ident

set dcc-block: sebbene il file di configurazione di esempio raccomandi di impostare 0 (turbo DCC), questo può causare l'interruzione prematura del trasferimento DCC. Se si useranno molto i trasferimenti DCC si setti a 1024.

Per concludere, ci si assicuri di aver rimosso il comando 'die' dalla configurazione (ce ne sono un paio nascosti in vari posti). Finito di impostare il file di configurazione ci si accerti di averlo rinominato con qualcosa di diverso da eggdrop.conf
Poi (se si è impostato il file di configurazione in locale) si carichi il file nella directory in cui è installato il bot.

3.3USO DI UN EGGDROP

Si può comandare un Eggdrop sia da telnet che tramite ctcp dcc chat.
Una volta stabilita la conessione col bot, verrà richiesta la password e poi, automaticamente, si verrà inseriti in party line (la maggiore area di chat).
Si possono, a questo punto, digitare i comandi dell'Eggdrop facendoli precedere da un punto (x es,. .help, .bots, .whom, ecc).
Tutto ciò che non è preceduto da un punto viene inviato agli altri utenti della party line, proprio come un messaggio inviato su un canale IRC.
Esistono anche altri canali sul bot (come le party line) su cui è possibile immettersi, tanto

che il bot assomiglia un po' ad un server IRC.

USO DELLA CONSOLE

L'Eggdrop ha un ampio sistema di aiuto interno.La prima cosa da fare, una volta aperta una sessione di chat in DCC col bot, è digitare .help per vedere la maggior parte dei comandi utilizzabili.
Per maggiori informazioni su un comando in particolare si può digitare .help seguito dal comando che interessa (es. .help adduser).
La sessione di chat in DCC non solo permette di parlare con gli altri utenti nella party line (e degli altri 'canali' interni) e usare i comandi Eggdrop, ma permette anche di controllare il bot.

Usando il comando .console, è possibile cambiare il tipo di informazioni chevengono mostrate, si può ad esempio scegliere di vedere i comandi usati dagli altri, di visualizzare i messaggi e i notice mandati al bot, i messaggi pubblici al canale, ecc.
Le impostazioni sulla console determinano anche su quale canale IRC si sta lavorando; se il bot è sui canali #donkeys e #horses, è possibile impostare la console su uno di questi.
Molti dei comandi dell'Eggdrop verranno applicati al canale corrente sulla console, ad esempio se la console è impostata sul canale #donkeys e si usa il comando .op Pippo, il bot opperà 'Pippo' su #donkeys.
Si può salvare il canale corrente sulla console usando .save

COMANDI DA CONSOLE

COMMANDS for “nome eggdrop”, eggdrop v1.6.15:
who away quit whom me
page match motd bots newpass
chat handle whoami echo strip
su trace fixcodes bottree vbottree
botinfo relay -host
For ops:
addlog console match whois
For botnet masters:
+bot botattr chhandle chpass +host
-bot link chaddr boot
unlink banner dccstat
For masters:
chattr save backup reload status
traffic uptime
+user +ignore comment binds ignores
-user -ignore dccstat debug rehash
restart module
For owners:
die simul loadmod unloadmod language
set tcl rehelp modules +lang
-lang +lsec -lsec lstat relang
ldump
All commands begin with '.', and all else goes to the party line.
Text starting with ',' is sent ONLY to bot-masters.
You can get help on individual commands: '.help <command>'
Extra help relating to loaded modules may be obtained by typing
'.help <module> module'. Possible modules include:
assoc channels console transfer irc
filesys notes seen server share
You may receive a list of commands provided by all loaded modules by
using '.help all'. If you only remember a part of the command's
name you are searching for, just use wildcards (e.g. '.help *bot*'),
and all matching help texts will be displayed.

COMANDI ATTRAVERSO MESSAGGI

L' Eggdrop ha un numero limitato di comandi attivabili attraverso messaggi, ma alcuni di questi, come i comandi op e ident, sono importanti.
Per la lista dei comandi via messaggio, si scriva /msg help.
Comandi addizionali via messaggio possono essere aggiunti con uno script Tcl.
Si presti molta attenzione nell'usare comandi via messaggio che includono la password: se si è soliti scrivere /msg nella finestra del canale del client IRC, si potrebbe accidentalmente inviare la password al canale dove essa è visibile a tutti!

COMANDI PUBBLICI

L' Eggdrop non ha comandi pubblici impostati (comandi che si scrivono sul canale) eccetto 'seen'.
Se ovviamente si è caricato il modulo seen (loadmodule seen nel file di configurazione) e il canale è impostato +seen, allora il bot risponderà al comando seen nel canale.
Ci sono diversi script Tcl che aggiungono comandi pubblici all'Eggdrop, ma molti di questi script sono insicuri (permettono ad utenti malevoli di avere il controllo del tuo Eggdrop).
Un buono script di comandi pubblici deve permettere solo agli utenti oppati di usare i comandi. Sebbene molti pensino che sia fico poter usare comandi pubblici senza essere oppati, questo rende il tutto poco sicuro.

IL FILE USER

Il file User dell'Eggdrop controlla quali utenti possono accedere al bot, e il livello di accesso che ogni utente ha. Il file User contiene anche la lista dei ban e la lista degli ignore.
La gestione dell'userfile è una delle cose fondamentali da imparare per saper usare effettivamente l'Eggdrop.
Quando si lancia per la prima volta il proprio Eggdrop e ci si presenta usando il comando hello o equivalente, si viene inseriti al primo posto ottenendo così tutti i privilegi dell'owner (se si è disabilitato learn-users, il comando hello verrà disattivato una volta che ci si sia presentati come owner).
Per vedere le entrate dell'userfile, si scriva .whois nella console.
Verrà visualizzata una cosa simile alla seguente:

HANDLE PASS NOTES FLAGS LAST

ProprioNick yes No fjmnoptx 19:47(PartyLine)

HOSTS: *!mynick@*.nice.net, *!mynick@207.324.333.*

Le informazioni qui sopra mostrano l'handle dell'utente (l'handle è semplicemente il nickname dell'utente sul bot, quello con cui il bot lo ha conosciuto e da cui lo riconosce), se ha o no la password impostata, quanti messaggi ha, tutti i suoi canali e flags, quando e dove è stati visti dal bot l'ultima volta e le proprie hostmasks. Di seguito si apprenderà come modificare l'userfile.

VISUALIZZAZIONE DI TUTTI GLI UTENTI

Per vedere tutti gli utenti nell'userfile del bot, si scriva .match * 9999.
Si avrà modo di vedere qualcosa di simile a quanto mostrato sopra per ogni utente.
Aggiungere/rimuovere utenti
Ci sono tre modi in cui un utente può venire aggiunto al bot.
Se learn-users è abilitato nel file di configurazione, ognuno può mandare il messaggio col

comando hello al bot e il bot lo aggiungerà con la flag di default (come impostata nel file di configurazione alla voce default-flags).
Comunque si può aggiungere un utente usando anche il comando .adduser o .+user nella console.
Se l'utente che si vuol aggiungere è in uno dei canali del bot, allora il comando .adduser è il più conveniente.
Ci si assicuri che che il canale dell'utente sia il canale corrente sulla console (si veda di nuovo Uso della console), poi si scriva .adduser <nick>

dove nick è il nickname dell'utente che si vuol addare. L'utente sarà aggiunto all'userfile con la flag di default, la sua hostmask verrà aggiunta automaticamente.
Il comando .+user può essere usato quando una persona che vuol aggiungere non è in IRC.
Si scriva .+user con nickname e hostmask. Una volta che un utente è aggiunto deve impostare la password con il comando pass (si ricordi all'utente di scrivere / msg <pass> per impostare la password).

E' possibile anche impostare la password al posto suo con il comando .chpass (spiegato di seguito). Per rimuovere un utente dal bot, scrivere semplicemente .-user con nickname.

CAMBIARE LA PASSWORD DI UN UTENTE
Per impostare la password di un utente o cambiarla, si scriva .chpass
Si può non impostare affatto la password, scrivendo .chpass senza specificare la password

AGGIUNGERE/RIMUOVERE HOSTMASK
Il comando .+host permette di aggiungere una hostmask ad un utente: .+host hyena*!hyena@*.africa.net.
Per rimuovere una hostmask di un utente usa .-host .

LA BAN LIST
La ban list è una parte dell'userfile specificamente riservata per immagazzinare i ban.
I ban sono aggiunti ad una ban list interna al bot (chiamata anche enforced o permanent ban list) usando il comando .+ban seguito da <nick> o <host> e 0 se il ban è permanente.
Possono anche venir aggiunti automaticamente dal bot (ad esempio in risposta ad un flood) o da uno script Tcl.
I ban interni possono anche essere global (sono applicati su tutti i canali su cui è il bot) o channel-specific.
Un ban può essere permanente o estinguersi automaticamente dopo un certo periodo impostato, e può essere 'sticky' (il bot si assicurerà che il ban sia sempre attivo sul canale).
Nota che se si sta usando +dynamicbans nei settaggi di un canale, un ban settato nella lista interna dei ban del bot, verrà rimosso dopo i minuti di ban-time (come settato nel file di configurazione), ma rimarrà nella lista interna dei ban e verrà riattivato tutte le volte che qualcuno corrispondente al ban entra sul canale.
Se si sta utilizzando +dynamicbans e si vuole che il ban sia sempre attivo sul canale, si deve rendere il ban sticky.

VISUALIZZAZIONE DELLA LISTA DEI BAN

Per vedere tutti i global ban correnti e i channel ban (per il canale corrente della console) attivi, si scriva .bans. Per vedere sia i ban attivi che quelli inattivi si scriva .bans all.
La lista mostra anche i ban che sono attivi su un canale, ma non nella lista dei ban interna del bot (questo genere di ban sono preceduti da un asterisco).

AGGIUNGERE/RIMUOVERE I BAN

I ban global sono aggiunti con .+ban (ad es .+ban *!*lamer@*.isp.net). Si può aggiungere un ban specifico di canale usando .+ban <#channel>.
I ban aggiunti usando questi comandi saranno permanenti (rimarranno nella lista dei ban interna del bot finchè qualcuno non li rimuoverà manualmente). Per rimuovere un ban esistono due modi: utilizzare la banmask o il numero di riferimento.
La lista che esce con .bans all mostrerà infatti un numero di riferimento prima di ogni ban per cui, se si volesse rimuovere il ban numero 4, si scriverà dovrai scrivere .- ban 4.
Si tenga a mente che i numeri di riferimento dei ban cambiano in base al canale settato sulla console (per es se si scrive .console #horses e poi .bans all, poi .console #donkey, poi .bans all, i numeri di riferimento dei bans potrebbero essere diversi nelle due liste mostrate).
Per rimuovere un ban con la banmask, si scriva semplicemente .-ban seguito dalla mask.

FARE UN BAN STICKY

Puoi fare un ban sticky usando il comando .stick con il numero di riferimento o la banmask. Un ban sticky verrà riattivato dal bot se qualcuno lo rimuove dal canale. Si ricordi di usare l'aiuto interno dell'Eggdrop (.help) per imparare di più sui comandi.
Una volta che hai preso confidenza con le funzioni e i comandi di base dell'Eggdrop guarda come potenziare il tuo Eggdrop.

IMPOSTAZIONI DI CANALE

Il modo in cui l'Eggdrop agisce e risponde a ciò che accade sul canale dipende dalle impostazioni del canale.Eggdrop ha molte impostazioni di canale già settati, si possono settare differenti impostazioni per ogni canale, permettendo estrema flessibilità.
Una volta creato il file di configurazione dell'Eggdrop e aggiunte le entrate per ogni canale in cui il bot risiederà, dci si dovrebbe imbattere per la prima volta nelle impostazioni dei canali.
Nella guida all'impostazione si ricordi ricordare di aver visto il seguente:
channel add #horses {
options
}
channel set #horses +option -option
Ci sono 2 battute di settaggio dei canali. La prima battuta,quella tra le parentesi graffe come un settaggio idle-kick e flood protection (x es flood-join, ecc.).
La seconda battuta è un' attivazione/disattivazione di un comando channel.
Queste opzioni riguardano comandi come autoop, dynamicbans, revenge, ecc.
Tali impostazioni sono precedute da un segno + o - per specificare rispettivamente se si vuole o no che l'impostazione sia attiva o inattiva. Per ogni ulteriore informazione su ciascuna impostazione channel e la sua funzione si faccia riferimento al file eggdrop.conf.dist del bot.

IMPOSTAZIONI DINAMICHE PER UN CANALE

Se si vuol aggiungere un canale al tuo bot o cambiare un' impostazione di un canale, non è necessario modificare il file di configurazione. L'Eggdrop ha dei comandi DCC preimpostati che permettono di aggiungere/rimuovere un canale e di cambiarne le impostazioni attraverso la console.
Per aggiungere un canale si scriva semplicemente .+chan <#channel>
Per rimuoverlo .-chan <#channel>
Le impostazioni di un canale vengono modificate usando il comando .chanset il quale funziona differentemente a secondache si stia impostando comandi come autoop, dynamicbans, ecc, o cambiando una opzione di canale come idle-kick.
Sotto ci sono alcuni esempi di utilizzo del comando .chanset:
.chanset <#channel> +enforcebans
attiva l'opzione enforceban
.chanset <#channel> -dynamicbans +autoop
disattiva l'enforceban e attiva l'autoop
.chanset <#channel> chanmode +sntk green
cambia le impostazione del canale in "+sntk green"
.chanset <#channel> idle-kick 60
imposterà il settaggio idle-kick a 60

IL CHANFILE

Dal momento che il bot non può modificare da solo il file di configurazione, i canali aggiunti col comando .+chan e i settaggi di canale modificati con .chanset hanno bisogno di essere immagazzinati in un file speciale chiamato chanfile.
Si dovrebbe ricordare di aver determinato il chanfile nel file di configurazione del file (per es set chanfile "mybot.chan").
Il chanfile assicura che ogni cambiamento che impostato usando comandi in DCC verranno ricordati anche se il bot viene disattivato e riavviato.
Un inconveniente nell'avere il chanfile è dato dal fatto che, quando il bot parte, per prima cosa legge dal file di configurazione, poi dall chanfile.Ogni impostazione di canale settata nel file di configurazione verrà sovrascritta da quelli del chanfile-se si fanno cambiamenti nel file di configurazione, questi non avranno effetto.
Per cambiare le impostazioni di un canale si dovranno usare i settaggi in DCC. Per questo può accadere spesso nelle impostazioni di un canale che esse non corrispondano a quelle specificate nel file di configurazione.
Per tale motivo alcuni scelgono di non aggiungere canali nel file di configurazione, usando invece esclusivamente i comandi in DCC per aggiungere canali e modificare le loro impostazioni.
Si può decidere di non avere chanfile scrivendo "" nel file di configurazione del bot (set chanfile ""). Questo permetterà di fare tutti i cambiamenti delle impostazioni di un canale sul file di configurazione, ma ogni cambiamento fatto usando comandi DCC non verrà ricordato dal bot.

La prima cosa da fare quando si accede al proprio bot per la prima volta e' presentarsi
/msg botnick hello (cosi' ci riconosce e ci chiedera' di settarci la password)
/msg botnick pass lamiapass

COMANDO SINTASSI E DESCRIZIONE

<OWNERS>
.chnick vecchionick nuovonick (Cambia il nick dell'utente specificato all'interno della party-line)
.chpass nick nuovapass (Cambia la password di un utente)
.chattr nickname [attributo] (Permette di variare le flag di un utente)
.comment nickname [commento] (Permette di creare o variare i commenti per l'utente specificato. Attenzione, i commenti sono visibili dai soli master o owner del bot
.msg nickname [messaggio] (Invia un messaggio ad un utente specifico)
.chemail nickname [e-mail] (Setta l'e-mail nell'apposito campo nelle informazioni utente
.adduser nickname (Aggiunge alla lista un utente che è presente in canale con una hostmask di livello 3 (*!userid@*.dominio.it))
.chaddr bot address:numero porta (Cambia l'address di un bot per linkare il proprio alla botnet)
.+user nickname hostmask (Aggiunge un utente al database con l'hostmask definita fra i parametri)
.-user nickname (Elimina l'utente dal data base)
.+host nickname hostmask (Aggiunge una nuova hostmask all'utente predefinito)
.-host nickname hostmask (Elimina una hostmask dall'utente predefinito)
.chinfo nickname [info-line] (Varia le informazioni per l'utente predefinito)
.save (Nei bot eggdrop il salvataggio del data base utenti avviene periodicamente, se si effettua una modifica questo può essere forzato per rendere effettivi i cambiamenti)
.binds [tipo] (Mostra i bindings tcl. Questi non sono altro che degli "alias" con i quali lanciare i comandi tcl
.reset [canale] (Resetta le informazioni per il canale specificato)
.link bot (Esegue il link alla botnet)
.unlink nomebot (Rimuove il link del bot specificato dalla botnet)
.trace bot (Invia un pacchetto di tracciatura al bot specificato mostrando il percorso fatto)

.banner messaggio (Invia un messaggio a tutti gli utenti attualmente connessi al bot, solitamente questo comando viene usato per annunciare uno shutdown)
.boot nickname [messaggio] (Esegue l'espulsione di un utente dalla party-line)
.boot nickname@bot [messaggio] (Se i bot sono configurati per i boot remoti esegue un boot remoto dell'utente all'interno della botnet)
.relay botname (Esegue la riconnessione al bot specificato via telnet finché non verrà specificato il comando "*bye*" in party-line)
.flush bot (Esegue la pulizia del resync-buffer del bot nel caso le sue liste siano condivise con altri bot della botnet)
.status (Restituisce lo stato di funzionamento del bot in forma breve)
.dccstat (Visualizza in formato tabulare le informazioni su tutte le connessioni attive)
.debug (Se configurato durante la fase di compilazione del sorgente c del bot restituisce un dump della memory allocation)
.set variabile [valore] (Cambia i parametri dei settaggi interni del bot)
.rehash (Inizializza il bot rileggendo il file init, da utilizzare ogni volta che si esegue un

cambiamento all'interno di questo file)
.reload (Esegue il ricaricamento del data base utenti)
.die (Esegue lo shutdown del bot)
.restart (Salva e ricarica la user list)
.dump testo (invia il testo specificato direttamente al server)
.console attributo (Permette di variare la console di sistema)
.+ignore nickname [hostmask] (Aggiunge l'utente alla ignore list)
.-ignore nickname [hostmask] (Rimuove l'utente dalla ignore list)
.ignores [chiavedi ricerca] (Visualizza il contenuto della ignore list tramite una chiave di ricerca)
.ignores all (Visualizza il contenuto della intera ignore list)
.jump [irc srv] (Esegue il cambio di server)
.assoc [numero canale nome] (Associa il nome ad uno dei canali della party-line)
.assoc numero canale (Rimuove il nome canale)

<OPERATORI>
.who [bot] (Restituisce la lista degli utenti attualmente connessi al bot)
.whois nickname (Mostra le informazioni per l'utente specificato)
.match +o [[start] limit]] (Esegue una ricerca nel data base utenti in base alla flag specificata)
.bots (Mostra la lista dei bots attualmente connessi alla botnet)
.bottree (Mostra la lista dei bots attualmente connessi alla botnet in formato ad albero)
.notes index (Mostra la lista dei messaggi a voi destinati attualmente in archivio)
.notes read (Lettura di uno o più messaggi)
.notes erase (Cancellazione di uno o più messaggi)
.echo [ON/OFF] (Permette di attivare o disattivare l'echo locale)
.+ban hostmask[commento] (Permette di aggiungere un ban permantente alla lista)
.-ban hostmask [commento] (Permette di rimuovere un ban permanente alla lista)
.bans [chiave di ricerca] (Visualizza la lista dei ban tramite una chiave di ricerca)
.bans all (Visualizza l'intera lista dei ban)
.resetbans (Resetta la ban list del canale)
.op nickname (Oppa l'utente)
.deop nickname (Deoppa l'utente)
.topic [messaggio] (Setta il topic del canale sul quale è la propria console)
.say [messaggio] (Permette al bot di "parlare" sul canale nel quale la console dell'utente è settata)
.act [messaggio] (Esegue una action sul canale nel quale la console dell'utente è settata)

.motd (Rivisualizza il Message Of The Day)
.addlog messaggio (Permette di lasciare commenti nel log del bot)
.invite nickname (Nel caso il canale sul quale la console utente è settata sia +i questo comando fa si che il bot vi inviti)
.nick nickname (Cambia il vostro nick all'interno della party-line)
.away [messaggio] (Permette si settarvi away in party-line. Una volta che farete qualcosa l'away verrà rimosso automaticamente)
.me (Esegue una acton in party-line)
.note nickname [messaggio] (Permette di lasciare un messaggio ad un determinato utente)
.files (Se il flag opportuno è settato, fa accedere all'area file del bot)
.newpass nuovapassword (Permette di variare la password di accesso)

.console attributo (Permette di variare la console di sistema)
.email emailaddress (Inserisce un indiizzo e-mail nelle vostre informazioni utente)
.quit msg (Uscita dalla party-line)
.servers (Mostra la lista dei servers IRC alla quale il bot si collega)
.channel [nome-canale] (Mostra gli utenti attualmente sul canale specificato, a condizione che il bot sia presente su quel canale)
.kick nickname [messaggio] (Esegue il kick dell'utente)
.chat off (Uscita dalla party-line, si può comunque continuare a comandare il bot via dcc)
.chat on (Ritorno al canale di default nella party-line (channel 0))
.chat numero canale o nome (Questa opzione consente di creare un nuovo canale nella party-line)
.info [info-line] (Setta le informazioni per l'utente)
.info (Rimuove le informazioni per l'utente).

3.4 LINKARE + EGGDROP INSIEME

Per gestire meglio un canale conviene avere più eggdrop e linkarli in modo che ognuno sharerà all'altro i propri dati utenti.

Flag	Descrizione
s	share (il bot condivide il database utenti, aggressivamente)
p	share (il bot condivide il database utenti, passivamente)
g	global share (condividi le informazioni su tutti i canali)
h	hub (il bot è automaticamente linkato con la massima priorità)
a	alternate (il bot è automaticamente linkato se non ci sono hub al quale può linkare)
l	leaf (il bot non è abilitato a linkare con altri bots)
r	reject (il bot non è rifiutato sulla botnet)
i	isolate (isola la partyline)

Ora, con un esempio pratico consideriamo che il bot chiamato "Bot-Hub" sia il bot principale al quale tutti gli altri debbano collegarsi;
Questi settaggi vanno impostati su ogni bot che non sia Bot-Hub, che si vuole linki a Bot-Hub:

.+bot Bot-Hub ip-bot-hub:porta-telnet/porta-relay
.+host Bot-Hub *!ident@ip-bot-hub
.botattr Bot-Hub +ghp
.chattr Bot-Hub +foN

Legenda:

Bot-Hub	è il nick del bot
ip-bot-hub	è l'ip della shell (o del vhost) sul quale si trova il bot
porta-telnet	è la porta al quale gli altri bots linkano
porta-relay	è la porta che permette la connessione in partyline tramite un bot ad un'altro bot della botnet (per info vedi .relay)

*!ident@ip-bot-hub	è il semplice host che il bot ha su irc, conviene sempre aggiungere anche l'hostname risolto ad ip

le botflag "**hp**" dice al bot che deve scaricare il database utente da Bot-Hub e deve linkarsi a lui in automatico, le userflag "**fo**" servono a dichiarare Bot-Hub come operatore in tutti i canali. Un caso speciale è la botflag "**N**" che però è settata nella parte dedicata alle userflag, dichiara che quel determinato eggdrop è abilitato e supporta le tcl netbots.

Questi settaggi vanno impostati su Bot-Hub in modo da accettare le connessioni degli altri bots e linkare:

.+bot Bot-Pas ip-bot-pas:porta-telnet/porta-relay
.+host Bot-Pas *!ident@ip-bot-pas
.botattr Bot-Pas +gs
.chattr Bot-Pas +foN

In questo modo, ora, Bot-Hub vede il bot passivo Bot-Pas come bot abilitato a linkare a lui; le botflag "**gs**" dicono all'hub di effettuare una condivisione aggressiva del database utenti.

P.S. Conviene sempre settare prima i dati su Bot-Hub e poi su Bot-Pas in modo che il link sia automatico e senza problemi.

4.PSYBNC

Questo software è utilizzato per mantenere una connessione ad IRC.

Cosa può fare psyBNC ?

1) Resta connesso quando ti disconnetti
2) Amministrazione completa in linea.
3) Supporto VHOSTS. Si possono usare VHOSTS su bouncers collegati, se l'amministratore ha dichiarato il link al tuo bouncer come relaylink.
4) Si possono impostare bans sul client
5) Si possono impostare utenti che possono opparsi dal bouncer con password e hostname corrispondenti.
6) Si possono impostare Hosts e Passwords ai client, da cui il bounce potrebbe opparsi, se esso dovesse disconnettersi o soltanto si volesse automatizzare il processo anche quando sei connesso. Inoltre psyBNC consente di connettersi automaticamente ad un eggdrop e di opparsi da questo.
7) Consente una partyline sul bounce (un canale &partyline) o di conversare con utenti specifici sul bounce ($ più login-utente), anche su bouncer collegati.
8) Registrazione completa di messaggi e traffico.
9) Consente di collegarsi ad altri bouncers e di condividerne la partyline.
10) Consente connessioni multiple di un Client a servers e reti differenti
11) Criptatura password Blowfish integrata.
12) Collegamento automatico di un utente a bot eggdrop via DCC. Il traffico da/al bot verrà gestito tramite una query al nick: "-nomebot".
13) Criptatura connessioni e conversazioni.

4.1INSTALLAZIONE

Scaricare il sorgente, decomprimerlo e digitare make.

Se si vuole cambiare la porta di ascolto (porta di default 31337) basta editare il file psybnc.conf altrimenti procediamo col il lancio del programma digitando ./psybnc

Una volta lanciato il programma bisogna collegarsi allo psybnc tramite la porta scelta da voi utilizzando il vostro client irc.

4.2 COMANDI

Comando	Funzione
/BHELP	Mostra un elenco dei comandi
/BWHO	Mostra l'elenco degli utenti del BNC
/ADDUSER login	Aggiunge un utente
/DELUSER login	Elimina un utente
/PASSWORD newpass	Cambia la password per in network corr.
/JUMP	Jumpa al server successivo in lista
/ADDSERVER	Aggiunge un server alla lista
/DELSERVER	Elimina un server dalla lista
/LISTSERVERS	Mostra la lista dei server
/REHASH	Rilancia il BNC e lo rehasha
/BQUIT	Chiude tutte le connessioni a IRC
/BCONNECT	Ritenta una connessione all'ultimo server
/SETAWAYNICK nick	Imposta un nick per il periodo di away
/SETAWAYNICK	Disabilita il cambio di nick per l'away
/SETLEAVEMSG message	Imposta un messaggio di uscita per l'away
/SETLEAVEMSG	Disabilita il messaggio di uscita per l'away
/VHOST	Aggiunge un virtual host alla lista del BNC
/ADDOP login	Aggiunge un utente alla lista degli op
/DELOP login	Elimina un utente dalla lista degli op
/ADDBAN *!*@*	Aggiunge un ban
/DELBAN *!*@*	Elimina un ban
/PLAYPRIVATELOG	Mostra i messaggi di testo
/ERASEPRIVATELOG	Elimina i messaggi di testo salvati
/NAMEBOUNCER name	Assegna un nick al vostro BNC
/LINKTO :host:port	Aggiunge un host e una porta di un altro psyBNC a cui linkarsi
/LINKFROM name :host:port	Aggiunge un bouncer a cui linkarsi
/LISTLINKS	Mostra la lista dei links attivi
/DELLINK number	Elimina un link attraverso il numero espresso nella LISTLINK

digitare /quote bhelp nel psybnc per una lista completa

5.IPV6

IPV6 è un nuovo protocollo che lavora al livello 3 della pila protocollare ISO/OSI (descritto nell'RFC 2460, reperibile.....)
Progettato per sostituire il protocollo attualmente in uso, ovvero ipv4 (descritto nell'RFC 760, reperibile)in quanto essendo in uso da piu` di venti anni si va sempre piano paino alla progressiva diminuizione di indirizzi IP disponibili per le nuove macchine che devono collegarsi in Rete.

Inoltre ha anche altre innovazioni, qauli:

- Capacità di instradamento e indirizzamento espanso.
- Dimensione dell'indirizzo IP da 32 a 128 bit per supportare più livelli gerarchici di indirizzamento ed un numero molto più grande di nodi indirizzabili.
- Semplificazione dell'Header.
- Gestione per il controllo di flussi di traffico.
- Ottimizzazione delle funzioni di controllo.
- Indirizzo anycast.
- Capacità di autenticazione e privacy.
- Allineamento a 64 bit.
- Possibilità di espansioni future per il protocollo.

IPv6 ingloba determinate caratteristiche studiate appositamente per la fase di migrazione da Ipv4, infatti gli indirizzi IPv6 possono essere ricavati dagli indirizzi IPv4, e si possono creare tunnel IPv6 su reti IPv4.

Il tunneling permette quindi di trasportare il traffico Ipv6 usando il routing Ipv4.
Quindi gli Host e i Router dual stack (IPv4/IPv6) utilizzano il tunnel per instradare il traffico Ipv6 su porzioni di rete che hanno unicamente IPv4

5.1 CREAZIONE DI UN TUNNEL IPV6

- Controllare che la nostra macchina supporti ipv6, digitando lsmod.
- Se non è caricato, procediamo tramite il comando:
  modprobe ipv6
  Se ci ritorna "can't locate module ipv6" probabilmente il nostro kernel non è abilitato a tale protocollo e dovremo così procedere alla ricompilazione del Kernel

- Iscriversi ad un tunnel broker

NOTA: per semplificare la guida riportiamo come esempio un nome di una società Tunnel Broker
https://tb.ngnet.it

- Scaricare e installare un programma di tunnel:
  NOTA: per semplificare la guida riportiamo un esempio preso da:
- http://www.azzurra.org/download/6tunnel.tar.gz

- Effettuare il login al Tunnel Broker.
  NOTA: per semplificare la guida riportiamo un esempio di uno script preso da:
  http://www.azzurra.org/download/cselt1.0.tar.gz
  e poi procediamo a configurarlo con gli appositi dati personali.
- Ora attiviamo il tunnel con la seguente sintassi:
  ./6tunnel [porta_sorgente] [ipv6_del_broker] [porta_remota]
- Non ci resta che lanciare il nostro client irc e farlo connettere al nostro localhost.

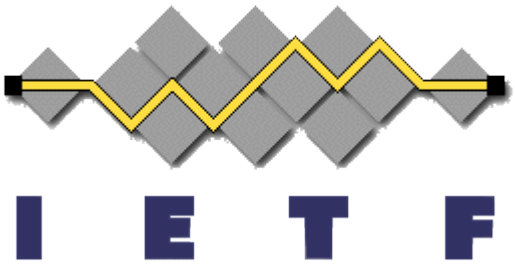
5.2 COLLEGAMENTO DIRETTO AD UN SERVER IPV6

Se invece abbiamo client irc con supporto ipv6, una volta attivato il sit0 possiamo collegarci direttamente al server irc senza creare un tunnel ipv6

RFC

Request for Comments

1459



Richiesta di commenti;
Lo standard RFC descrive il Protocollo Internet (Internet Protocol, IP) e costituisce una direttiva consigliata dalla Internet Engineering Task Force che definisce l'architettura IP globale.
Il sito ufficilale della Internet Engineering Task Force è http://www.ietf.org
Il documento che seguirà è l'RFC 1459 ufficiale depositato nell'archivio della Internet Engineering Task Force, disponibile all'indirizzo:
http://www.ietf.org/rfc/rfc1459.txt?number=1459

Network Working Group J. Oikarinen

Request for Comments: 1459 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 4
1.1 Servers 4
1.2 Clients 5
1.2.1 Operators 5
1.3 Channels 5
1.3.1 Channel Operators 6
2. THE IRC SPECIFICATION 7
2.1 Overview 7
2.2 Character codes 7
2.3 Messages 7
2.3.1 Message format in 'pseudo' BNF 8
2.4 Numeric replies 10
3. IRC Concepts 10
3.1 One-to-one communication 10
3.2 One-to-many 11
3.2.1 To a list 11
3.2.2 To a group (channel) 11
3.2.3 To a host/server mask 12
3.3 One to all 12

3.3.1 Client to Client 12
3.3.2 Clients to Server 12
3.3.3 Server to Server 12
4. MESSAGE DETAILS 13
4.1 Connection Registration 13

4.1.1 Password message 14
4.1.2 Nickname message 14
4.1.3 User message 15
4.1.4 Server message 16
4.1.5 Operator message 17
4.1.6 Quit message 17
4.1.7 Server Quit message 18
4.2 Channel operations 19
4.2.1 Join message 19
4.2.2 Part message 20
4.2.3 Mode message 21
4.2.3.1 Channel modes 21
4.2.3.2 User modes 22
4.2.4 Topic message 23
4.2.5 Names message 24
4.2.6 List message 24
4.2.7 Invite message 25
4.2.8 Kick message 25
4.3 Server queries and commands 26
4.3.1 Version message 26
4.3.2 Stats message 27
4.3.3 Links message 28
4.3.4 Time message 29
4.3.5 Connect message 29
4.3.6 Trace message 30
4.3.7 Admin message 31
4.3.8 Info message 31
4.4 Sending messages 32
4.4.1 Private messages 32
4.4.2 Notice messages 33
4.5 User-based queries 33
4.5.1 Who query 33
4.5.2 Whois query 34
4.5.3 Whowas message 35
4.6 Miscellaneous messages 35
4.6.1 Kill message 36
4.6.2 Ping message 37
4.6.3 Pong message 37
4.6.4 Error message 38
5. OPTIONAL MESSAGES 38
5.1 Away message 38
5.2 Rehash command 39
5.3 Restart command 39

5.4 Summon message 40
5.5 Users message 40
5.6 Operwall command 41
5.7 Userhost message 42
5.8 Ison message 42
6. REPLIES 43
6.1 Error Replies 43
6.2 Command responses 48
6.3 Reserved numerics 56

7. Client and server authentication 56
8. Current Implementations Details 56
8.1 Network protocol: TCP 57
8.1.1 Support of Unix sockets 57
8.2 Command Parsing 57
8.3 Message delivery 57
8.4 Connection 'Liveness' 58
8.5 Establishing a server-client connection 58
8.6 Establishing a server-server connection 58
8.6.1 State information exchange when connecting 59
8.7 Terminating server-client connections 59
8.8 Terminating server-server connections 59
8.9 Tracking nickname changes 60
8.10 Flood control of clients 60
8.11 Non-blocking lookups 61
8.11.1 Hostname (DNS) lookups 61
8.11.2 Username (Ident) lookups 61
8.12 Configuration file 61
8.12.1 Allowing clients to connect 62
8.12.2 Operators 62
8.12.3 Allowing servers to connect 62
8.12.4 Administrivia 63
8.13 Channel membership 63
9. Current problems 63
9.1 Scalability 63
9.2 Labels 63
9.2.1 Nicknames 63
9.2.2 Channels 64
9.2.3 Servers 64
9.3 Algorithms 64
10. Support and availability 64
11. Security Considerations 65
12. Authors' Addresses 65

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.

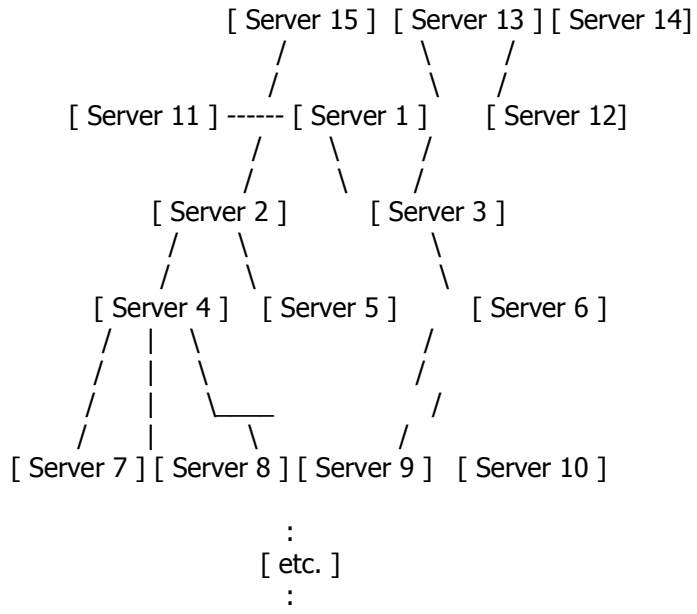
The IRC protocol has been developed on systems using the TCP/IP network protocol, although there is no requirement that this remain the only sphere in which it operates.

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>    ::= '-' | '[' | ']' | '\' | '`' | '^' | '{' | '}'
```



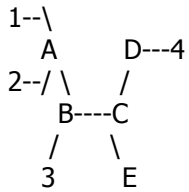
```
<nonwhite>   ::= <any 8bit code except SPACE (0x20), NUL (0x0), CR
                 (0xd), and LF (0xa)>
```

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

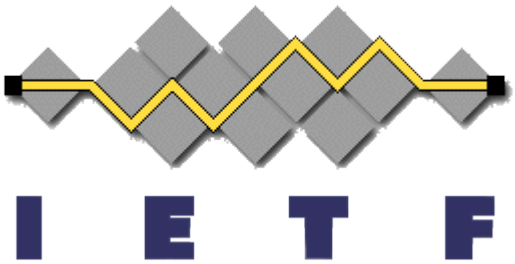
Watsonia, Victoria 3087
Australia

Email: avalon@coombs.anu.edu.au

RFC

Request for Comments

1459



Richiesta di commenti;
Lo standard RFC descrive il Protocollo Internet (Internet Protocol, IP) e costituisce una direttiva consigliata dalla Internet Engineering Task Force che definisce l'architettura IP globale.
Il sito ufficilale della Internet Engineering Task Force è http://www.ietf.org
Il documento che seguirà è una traduzione/adattazione dell'RFC ufficiale depositato nell'archivio della Internet Engineering Task Force, disponibile all'indirizzo:
http://www.ietf.org/rfc/rfc1459.txt?number=1459

Il protocollo IRC fu sviluppato da quando fu implementato per la prima volta come mezzo in grado di

permettere agli utenti di parlare tra loro su un BBS (BBS: Bullettin Board System. è un sistema a gestione privata al quale ci si può collegare tramite linea telefonica e che mette a disposizione programmi, dati, aree di messaggi etc.. nell'ambito della cosiddetta telematica amatoriale.- N.d.T.). Attualmente esso sostiene un network mondiale di server e clients, e si sta espandendo per far fronte alla crescente espansione dell'utenza. Negli ultimi 2 anni, il numero medio di utenti connessi al network IRC principale si è decuplicato. Il protocollo IRC è un protocollo, basato su stringhe di testo, il cui interlocutore più semplice consiste in un programma capace di stabilire una connessione col server.

Indice:

* 1. INTRODUZIONE
    - o 1.1 I Servers
    - o 1.2 I Clients
        - + 1.2.1 Gli Operatori
    - o 1.3 I Canali
        - + 1.3.1 Gli Operatori di canale
* 2. SPECIFICHE IRC
    - o 2.1 Generalità
    - o 2.2 Codici dei caratteri
    - o 2.3 Messaggi
        - + 2.3.1 Formato dei messaggi in 'pseudo' BNF
    - o 2.4 Risposte numeriche
* 3. CONCETTI DI IRC
    - o 3.1 La Comunicazione uno a uno
    - o 3.2 La Comunicazione Uno a molti
        - + 3.2.1 Verso una lista
        - + 3.2.2 Verso un gruppo (canale)
        - + 3.2.3 Verso un host/server mask
    - o 3.3 Uno a tutti
        - + 3.3.1 Comunicazione Client-Client
        - + 3.3.2 Comunicazione Client-Server
        - + 3.3.3 Comunicazione Server-Server
* 4. INFORMAZIONI DETTAGLIATE SUI SINGOLI MESSAGGI
    - o 4.1 Registrazione di connessione
        - + 4.1.1 Messaggio Password
        - + 4.1.2 Messaggio Nickname
        - + 4.1.3 Messaggio User
        - + 4.1.4 Messaggio Server
        - + 4.1.5 Messaggio Operator
        - + 4.1.6 Messaggio Quit
        - + 4.1.7 Messaggio Server Quit
    - o 4.2 Operazioni sul canale
        - + 4.2.1 Messaggio Join
        - + 4.2.2 Messaggio Part
        - + 4.2.3 Messaggio Mode
            - # 4.2.3.1 Modi dei canali
            - # 4.2.3.2 Modi dell'utente
        - + 4.2.4 Messaggio Topic
        - + 4.2.5 Messaggio Names
        - + 4.2.6 Messaggio List
        - + 4.2.7 Messaggio Invite
        - + 4.2.8 Messaggio Kick
    - o 4.3 Comandi e Richieste di informazioni sul server
        - + 4.3.1 Messaggio Version
        - + 4.3.2 Messaggio Stats
        - + 4.3.3 Messaggio Links
        - + 4.3.4 Messaggio Time
        - + 4.3.5 Messaggio Connect
        - + 4.3.6 Messaggio Trace

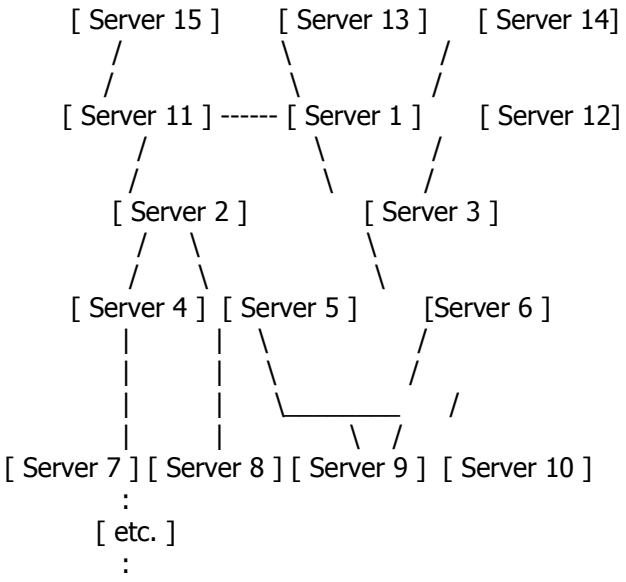
        + 4.3.7 Messaggio Admin
        + 4.3.8 Messaggio Info
    o 4.4 Invio di messaggi
        + 4.4.1 Messaggi Privati
        + 4.4.2 Messaggi Notice
    o 4.5 Richieste di informazioni sull'utente
        + 4.5.1 Richiesta Who
        + 4.5.2 Richiesta Whois
        + 4.5.3 Messaggio Whowas
    o 4.6 Altri messaggi
        + 4.6.1 Messaggio Kill
        + 4.6.2 Messaggio Ping
        + 4.6.3 Messaggio Pong
        + 4.6.4 Messaggio Error
* 5. MESSAGGI FACOLTATIVI
    o 5.1 Messaggio Away
    o 5.2 Messaggio Rehash
    o 5.3 Messaggio Restart
    o 5.4 Messaggio Summon
    o 5.5 Messaggio Users
    o 5.6 Messaggio Operwall
    o 5.7 Messaggio Userhost
    o 5.8 Messaggio ISon
* 6. RISPOSTE
    o 6.1 Risposte Error
    o 6.2 Replica Command
    o 6.3 Risposte numeriche riservate
* 7. AUTENTICAZIONE DI CLIENT E SERVER
* 8. DETTAGLI SULLE ATTUALI IMPLEMENTAZIONI
    o 8.1 Protocollo di rete: TCP
        + 8.1.1 Supporto per Unix sockets
    o 8.2 Analisi dei comandi
    o 8.3 Invio del messaggio
    o 8.4 Connessione 'Liveness'
    o 8.5 Effettuazione di una connessione server-client
    o 8.6 Effettuazione di una connessione server-server
        + 8.6.1 Scambio di informazioni di stato alla connessione
    o 8.7 Chiusura della connessione server-client
    o 8.8 Chiusura della connessione server-server
    o 8.9 Tracciamento dei cambi di nickname
    o 8.10 Controllo del flood dei clients
    o 8.11 Ricerche anti-blocco
        + 8.11.1 Ricerca dell'Hostname (DNS)
        + 8.11.2 Ricerca dell'Username (Ident)
    o 8.12 File di configurazione
        + 8.12.1 Autorizzazione dei client alla connessione
        + 8.12.2 Operatori
        + 8.12.3 Autorizzazione dei server alla connessione
        + 8.12.4 Informazioni accessorie sugli Amministratori del server
    o 8.13 Partecipazione ai Canali
* 9. PROBLEMI CORRENTI
    o 9.1 Adattamento a situazioni di dimensioni rilevanti
    o 9.2 Etichette
        + 9.2.1 Nicknames
        + 9.2.2 Canali
        + 9.2.3 Servers
    o 9.3 Algoritmi

1. INTRODUZIONE

Il protocollo IRC (Internet Relay Chat) è stato progettato e costruito, con un lavoro durato diversi anni, per la realizzazione di scambi di messaggi scritti in tempo reale. Questo documento descrive il protocollo di IRC attualmente in uso. (2) Il protocollo IRC è stato sviluppato su un sistema che usa il protocollo di rete TCP/IP, senza nessuna necessità, tuttavia, che questa rimanga il suo solo ambito di utilizzo. L'IRC stesso è un sistema di teleconferenza che (mediante l'uso del modello client-server) è stato adattato per lavorare su molte macchine di modelli diversi. Un tipico sistema comporta un singolo processo (il server) che costituisce un punto centrale al quale i client (o altri server) possono connettersi, realizzando i necessari invio e distribuzione contemporanea su più utenti del messaggio ed altre funzioni.

1.1 I Servers

Il server forma la spina dorsale di IRC, fornendo un punto al quale i client possono connettersi per parlarsi gli uni con gli altri, ed un punto di connessione per altri server, formando così una rete IRC. L'unica configurazione di rete permessa ai server di IRC è quella di un albero [vedi Fig. 1] dove ogni server agisce come nodo centrale per il resto della rete che vede.



[Fig. 1. Configurazione di una rete di server IRC]

1.2 I Clients

Si definisce client qualsiasi cosa connessa ad un server che non sia un altro server. Ogni client si distingue dagli altri client per un nickname unico, della lunghezza massima di nove (9) caratteri. Si vedano le regole grammaticali del protocollo per sapere quali caratteri si possono e quali non si possono usare in un nickname. In aggiunta al nickname, tutti i server devono avere le seguenti informazioni su ogni client: il vero nome della macchina sulla quale il programma client viene eseguito, il nome dell'utente del client di quella macchina, ed il server al quale il client è connesso.

1.2.1 Gli Operatori

Per far sì che il lavoro di rete venga svolto in maniera sufficientemente ordinata, si permette ad una determinata classe di client (operatori) di compiere delle funzioni di manutenzione generale sulla rete. Malgrado i poteri attribuiti ad un operatore possano essere considerati "pericolosi", essi sono tuttavia necessari. Gli operatori dovrebbero essere in grado di compiere operazioni di base sulla rete, quali lo sconnettere e il riconnettere servers, per prevenire di un routing non efficiente sulla rete. Prendendo atto di

questa esigenza, il protocollo qui esposto prevede per gli operatori solo la capacità di compiere quelle operazioni. Si vedano le sezioni 4.1.7 (Messaggio Server Quit) e 4.3.5 (Messaggio Connect).

Uno dei più controversi, tra i poteri attribuiti agli operatori è la possibilità di rimuovere "con la forza" un utente dalla rete: gli operatori sono in grado, per esempio, di chiudere la connessione tra un qualsiasi client e il server. La giustificazione per questo è cosa critica, dal momento che un abuso di questa possibilità risulta sempre fastidioso e distruttivo. Per altri dettagli sul' argomento si veda la sezione 4.6.1 (Messaggio KILL).

1.3 I Canali

Un canale è un gruppo, dotato di nome, di uno o più client dove tutti i membri del gruppo ricevono i messaggi indirizzati a quel canale. Il canale è automaticamente creato quando il primo client vi si collega, e cessa di esistere quando l'ultimo client lo lascia. Durante il periodo di esistenza del canale ogni client può riferirsi a quel canale usando il nome dello stesso. I nomi dei canali sono stringhe (che iniziano con un "&" oppure con un "#") lunghe fino a 200 caratteri. A parte la necessità che il primo carattere sia "&" oppure "#", l'unica restrizione che sussiste per il nome del canale è che non contenga alcuno spazio (" "), un control G (^G o ASCII 7), o una virgola ("," la quale è considerata dal protocollo come separatore tra gli elementi di una lista).

Ci sono due tipi di canali ammessi da questo protocollo. Uno è un canale assegnato che è conosciuto da tutti i server che sono connessi alla rete. Questi canali sono contrassegnati dall' avere come primo carattere (un #, gli altri sono i cosiddetti canali locali, il cui nome è preceduto da un carattere & e che) si distinguono per essere accessibili solo ai client del server ove il canale esiste. [Ndr - Qui probabilmente il testo originale a nostra disposizione omette parte del periodo, ma, dal contesto, si può tentare una interpretazione che abbiamo inserito tra parentesi tonde] Questi canali sono distinti dal carattere iniziale "&". Sono disponibili inoltre diversi modi disponibili per alterare le caratteristiche dei singoli canali. Si veda la sezione 4.2.3 (Messaggio MODE) per maggiori dettagli sull'argomento.

Per creare un nuovo canale o entrare a far parte di un canale già esistente, un utente entrare (JOIN) nel canale. Se il canale non preesiste al join, il canale viene creato ed il creatore del canale diventa operatore su quel canale. Se invece il canale già esiste, la richiesta di join al canale viene onorata o meno dipendentemente dai "modes" in vigore al momento su quel canale. Per esempio se si tratta di un canale ad invito (+i), la richiesta verrà accolta solo se si è stati invitati da qualcuno.

Secondo il protocollo, un utente può accedere a diversi canali contemporaneamente, ma si raccomanda di non superare il limite di dieci (10) canali, un confine ampio, sia per gli esperti che per i novizi. Si veda la sezione 8.13 per maggiori informazioni su questo argomento. (3) Se qualche ramo della rete IRC si spezza a causa di uno split (perdita del collegamento) tra due servers, del canale, su ognuno dei due versanti della rete divisi dal punto di interruzione, faranno parte solamente quei client che sono connessi al server sul rispettivo versante dello split, cessando di farne parte su quello opposto allo split. Quando il collegamento è ripristinato, i servers, di nuovo connessi, si comunicano l'un l'altro la propria lista dei client presenti sul canale i "modes" di quel canale. Se il canale esiste su tutti e due i versanti i JOIN e i MODE vengono interpretati in una maniera inclusiva cosicchè i due versanti della nuova connessione si troveranno in accordo su quali client sono nel canale e qual'è l' assetto dei mode del canale.

1.3.1 Gli Operatori di canale

L' operatore di canale (detto anche "chop" o "chanop") su un dato canale può essere considerato come "proprietario" di quel canale. A causa di questo loro stato, gli operatori di canale sono dotati di certi poteri che li mettono in condizione di mantenere controllo e una forma di pulizia sul loro canale. Come proprietario di un canale, ad un operatore non si richiede di render ragione per le sue azioni, tuttavia se le sue azioni sono particolarmente antisociali o costituiscono in qualche modo abuso, potrebbe essere ragionevole chiedere ad un operatore IRC di intervenire,oppure che gli utenti semplicemente escano e formino un loro canale.

I soli comandi che possono essere usati dagli operatori di canale sono:

KICK - Espelle un client dal canale
MODE - Cambia il mode del canale
INVITE - Invita un client ad un canale ad invito (mode +i)

TOPIC - Cambia il topic del canale in mode +t

Un operatore di canale è identificato dal simbolo '@' posto a fianco del suo nickname ogniqualvolta esso è associato con un canale (per esempio risponde ai comandi NAMES, WHO e WHOIS).

2. SPECIFICHE IRC

2.1 Generalità
Il protocollo qui descritto realizza sia connessioni server-server che connessioni client-server. Va detto, comunque, che ci sono più restrizioni nelle connessioni client-server (che sono considerate inattendibili) che in quelle server-server.

2.2 Codici dei caratteri
Nessun set particolare di caratteri è specificato. Il protocollo è basato su un sistema di codici composti da otto (8) bits, vale a dire un ottetto (byte). Ogni messaggio può essere composto da un numero qualsiasi di questi ottetti, sebbene i valori di alcuni ottetti vengano usati come codici di controllo, che svolgono il ruolo di delimitatori dei messaggi. Indipendentemente dal fatto di essere un protocollo a 8 bits, i delimitatori dei messaggi e i comandi sono organizzati in maniera tale da rendere il protocollo maggiormente utilizzabile da terminali USASCII e da connessioni telnet.

A causa dell'origine scandinava di IRC, i caratteri {} e | sono considerati essere l'equivalente minuscolo dei caratteri [] e \. Questo problema crea una situazione critica quando si tratta di determinare l'equivalenza di due nickname.

2.3 Messaggi
I server e i client si scambiano messaggi che possono generare o meno una risposta. Se il messaggio contiene un comando valido, come descritto nelle prossime sezioni, il client dovrebbe aspettarsi una risposta coerente con le specifiche, ma non è consigliabile che attende la replica del server per un tempo illimitato, poichè le comunicazioni client-server e server-server sono di natura essenzialmente asincrona.

Ogni messaggio IRC può essere consistere fondamentalmente di tre parti: il prefisso (facoltativo), il comando, ed i parametri del comando (ve ne possono essere fino a 15). Il prefisso, il comando e tutti i parametri sono separati da uno (o più) caratteri "spazio" (ASCII: 0x20).

La presenza di un prefisso è indicata con un singolo carattere due punti (":"), (ASCII: 0x3b), in prima posizione. Non ci devono essere lacune (spazi bianchi) tra i due punti ed il prefisso.

Il prefisso è usato dai server per specificare la vera origine del messaggio. Se manca il prefisso, si assume che il messaggio abbia avuto origine dalla connessione sulla quale è stato ricevuto. Non è necessario che un client premetta un prefisso quando inviano un messaggio: se usano un prefisso, l'unico prefisso valido è il nickname registrato ed associato con quel client. Se la sorgente identificata dal prefisso non può essere trovata nel database interno del server, oppure se la sorgente è registrata su un link differente da quello da cui arriva il messaggio, il server deve tacitamente ignorare il messaggio.

Il comando deve essere o un comando IRC valido, oppure deve essere un numero di tre (3) cifre rappresentato in codice ASCII. I messaggi IRC sono linee di caratteri che terminano sempre con una coppia CR-LF (Ritorno a capo - Salto di riga), e non devono eccedere i 512 caratteri in lunghezza, compresi tutti i caratteri inclusi i caratteri di coda CR-LF. Perciò il massimo numero di caratteri permesso per ogni riga di comando con gli eventuali parametri è di 510. Non c' è la possibilità di usare linee di continuazione per i messaggi. Si veda la sezione 7 per maggiori informazioni sulle implementazioni correnti.

2.3.1 Formato dei messaggi in 'pseudò BNF
I messaggi del protocollo devono essere estratti dal flusso continuo di ottetti. La soluzione attuale consiste nel designare due caratteri, CR e LF (ritorno a capo e salto di linea), come separatori di messaggi. I messaggi vuoti sono tacitamente ignorati, il che permette l'uso della sequenza CR-LF tra i messaggi senza ulteriori problemi.

Il messaggio estratto è scomposto ed analizzato nei componenti: (prefisso), (comando) e lista di parametri

uniti fra loro da componenti <intermedi> o <iniziali>.

Per tanto la rappresentazione BNF [Ndr - Backus Normal Form] di questo assetto è:

```
<messaggio> ::=
   [':' <prefisso> <SPAZIO> ] <Comando> <parametri> <crlf>
<prefisso> ::=
   <nome-del-server> | <nick> [ '!' <utente> ] [ '@' <macchina> ]
<Commando> ::=
   <lettera> { <lettera> } | <numero> <numero> <numero>
<SPAZIO> ::=
   ' ' { ' ' }
<parametri> ::=
   <SPAZIO> [ ':' <iniziali> | <intermedi> <parametri> ]
<intermedi> ::=
   < Qualsiasi sequenza *non vuota* di ottetti che non includa i caratteri SPAZIO, NUL (zero binario), CR, LF
e il cui primo carattere non può essere ':'>
<iniziali> ::=
   <Qualsiasi sequenza di ottetti, anche *vuota*, che non includa NUL o CR o LF>
<crlf> ::=
   CR LFG
```

Note:

1. <SPAZIO> consiste solo del(i) carattere(i) SPAZIO (0x20). Nota bene che la TABULAZIONE, e tutti gli altri caratteri di controllo sono considerati SPAZI NON BIANCHI.
2. Dopo aver estratto la lista di parametri, tutti i parametri sono equivalenti, non ha importanza che essi si possano associare con <intermedi> o <iniziali>. <Iniziali> è solamente un espediente sintattico per permettere lo SPAZIO tra i parametri.
3. Il fatto che CR e LF non possono apparire nel contesto dei parametri è solo un fatto dipendente dalla modalità di delimitazione del comando. Questo potrebbe cambiare in futuro.
4. Il carattere NUL non ha significato speciale nella composizione dei messaggi, e, in linea di principio, potrebbe essere parte di un parametro, causando però delle difficoltà nel normale trattamento delle stringhe in linguaggio "C". Questo è l' unico motivo per cui non è permesso l'uso del carattere NUL all'interno dei messaggi.
5. L'ultimo parametro può essere una stringa vuota.
6. L'uso del prefisso esteso (['!' <user>] ['@' <host>]) non deve essere usato nelle comunicazioni server-server, ed è previsto solo nell'ambito della comunicazione server- client in modo da fornire ai client maggiori informazioni utili riguardo la provenienza del messaggio, senza necessità di ulteriori domande.

Gran parte dei messaggi previsti dal protocollo specificano semantica e sintassi addizionali, dettata dalla loro posizione nella lista, per le stringhe dei parametri. Per esempio, molti comandi dei server assumono che il primo parametro dopo il comando è una lista di obbiettivi, così organizzata:

```
<obbiettivo> ::=
   <a> [ "," <obbiettivo> ]
<a> ::=
   <canale> | <utente> '@' <nome-del-server> | <nick> | <maschera>
<channel> ::=
   ('#' | '&') <stringa-di-caratteri>
<nome-del-server> ::=
   <macchina>
<macchina> ::=
   si veda l' RFC 952 [DNS:4] per dettagli sui nomi-macchina permessi
<nick> ::=
   <lettera> { <lettera> | <numero> | <carattere-speciale> }
<maschera> ::=
   ('#' | '$') <stringa-di-caratteri (contenente eventuali "*" e "?")>
<stringa-di-caratteri> ::=
   <qualsiasi sequenza di ottetti tranne SPACE, BELL, NUL, CR, LF e virgola (',')>
```

Ulteriori elementi sintattici dei parametri sono:

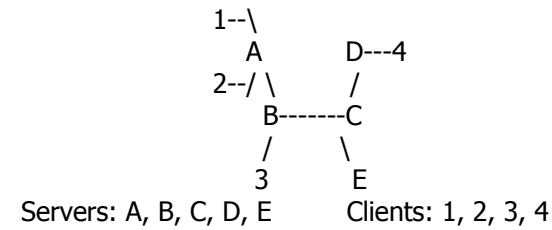
```
<utente> ::=
   <nonbianco> { <nonbianco> }
<lettera> ::=
   'a' ... 'z' | 'a' ... 'Z'
<numero> ::=
   '0' ... '9'
<speciale> ::=
   '-' | '[' | ']' | '\' | '`' | '^' | '{' | '}'
<nonbianco> ::=
   <ogni codice 8bit tranne SPACE (0x20), NUL (0x0), CR (0xd) e LF(0xa)>
```

2.4 Risposte numeriche

La maggior parte dei messaggi inviati al server generano una risposta di qualche sorta. La risposta più frequente è una risposta numerica, usata sia per risposte normali che di errore. La risposta numerica deve essere inviata come un messaggio consistente del prefisso del mittente, le tre cifre numeriche e la destinazione della risposta. Non è permesso che una risposta numerica abbia origine da un client; qualsiasi messaggio di questo genere ricevuto da un server viene tacitamente ignorato. Sotto tutti gli altri aspetti la risposta numerica è simile ad un normale messaggio, con la sola differenza che la parola-chiave è formata da tre cifre numeriche invece che da una stringa di lettere. Una lista delle possibili risposte è contenuta nella sezione 6.

3. CONCETTI DI IRC

Questa sezione è dedicata all' esposizione dei concetti su cui si basa l' organizzazione del protocollo IRC e come le implementazioni attualmente in uso inoltrino le differenti categorie di messaggi.



[Fig. 2. Esempio di una rete IRC di piccole dimensioni]

3.1 Comunicazione uno a uno

La comunicazione uno a uno è realizzata di solito dai clients, dal momento che la maggior parte del traffico server a server non è il risultato di server che parlino esclusivamente tra loro. Per dare ai client la possibilità di parlare loro, è necessario che tutti i server siano in grado di inviare un messaggio in una direzione precisa lungo la struttura ad albero della rete, in modo da poter raggiungere qualsiasi client. Il percorso di un messaggio inviato è quello più corto tra due punti qualsiasi della struttura.

Gli esempi che seguono fanno riferimento alla Figura 2.

Esempio 1:
   Un messaggio tra i client 1 e 2 è visto solo dal server A, il quale lo invia direttamente al client 2.
Esempio 2:
   Un messaggio tra i client 1 e 3 è visto dai server A e B, e dal client 3. A nessun altro server o client è permesso di vedere il messaggio.
Esempio 3:
   Un messaggio tra i client 2 e 4 è visto dai server A, B, C e D, e solamente dal client 4.

3.2 Uno a molti

Lo scopo principale di IRC è di realizzare un luogo pubblico di incontro nell' ambito del quale sia possibile, in maniera semplice ed efficiente, lo scambio di messaggi fra più persone (conversazione uno a molti).

A questo proposito IRC offre diversi mezzi, ognuno dei quali è orientato al raggiungimento di uno specifico scopo.

3.2.1 Ad una lista

Il tipo di conversazione uno-a-molti meno efficiente consiste nel parlare, attraverso i clients, ad una "lista" di utenti. Come questo avvenga è piuttosto banale: il client fornisce una lista di destinazioni alle quali il messaggio deve essere inviato ed il server lo "moltiplica", inviando delle copie separate del messaggio ad ogni destinazione data.

Questo sistema non è efficiente come quello della conversazione uno ad un gruppo, dal momento che la lista di destinazione viene considerata come composta da tante singole destinazioni e l'invio dei messaggi avviene senza controllare che non vengano inviati dei duplicati lungo i possibili percorsi.

3.2.2 Ad un gruppo (canale)

In IRC il canale ha il ruolo equivalente a quello di un gruppo ad invio multiplo; la sua esistenza è dinamica (sussistendo e/o cessando in funzione del fatto che gli utenti siano presenti o lascino il canale) e la conversazione in corso in un canale è inviata solo a quei server cui siano collegati utenti di un canale dato. Se ci sono più utenti sullo stesso server per lo stesso canale, il messaggio è inoltrato solamente una volta a quel server e poi inviato ad ognuno dei client del canale. Questa operazione viene poi ripetuta per ogni combinazione client-server finchè il messaggio originale non sia stato diffuso ed abbia raggiunto ogni membro del canale.

I seguenti esempi fanno riferimento alla figura 2.

Esempio 4:
Ogni canale con 1 solo client presente. I messaggi al canale vanno al server e da nessun altra parte.
Esempio 5:
2 client in un canale. Tutti i messaggi attraversano un percorso come se fossero messaggi privati tra i due client a prescindere dal canale.
Esempio 6:
Client 1, 2 e 3 dentro un canale. Tutti i messaggi al canale sono inviati a tutti i client e solo a quei server che devono essere attraversati dal messaggio come se esso fosse un messaggio privato ad un singolo client. Se il client 1 manda un messaggio, esso arriva al client 2 e poi, tramite il server B, al client 3.

3.2.3 Ad uno schema macchina-utente/server

Per fornire gli operatori di IRC di un qualche meccanismo per inviare messaggi ad un novero consistente di utenti, i messaggi stessi vengono corredati di "schemi macchina-utente e server". Questi messaggi sono inviati agli utenti per i quali le informazioni sulla macchina o sul server corrispondono a quelle dello schema. I messaggi vengono inviati solo alla locazione ove si trovano gli tenti, in un modo simile a quello usato per i canali.

3.3 Uno a tutti

Il tipo di messaggio uno a tutti è meglio denotato come un messaggio diffuso, inviato a tutti i client o/e a tutti i servers. In una rete di server e utenti di grandi dimensioni, un messaggio singolo può dar luogo ad un rilevante volume di traffico, nel momento in cui viene inviato attraverso la rete, per poter raggiungere tutte le destinazioni desiderate.

Per alcuni messaggi non c' è altra soluzione che diffonderli a tutti i servers, affinchè le informazioni di stato contenute da ogni server siano ragionevolmente coerenti tra i server stessi.

3.3.1 Client a Client

Non c' è alcuna categoria di messaggi per la quale, partendo da un messaggio singolo, derivi un messaggio

che sia inviato ad ogni altro client.

3.3.2 Client a Server
La maggior parte dei comandi che risultano nello scambio di informazioni di stato (quali appartenenza ad un canale, mode del canale, status dell' utente ecc.) deve, per default, essere inviata a tutti i server, e questa distribuzione non dovrebbe essere cambiata dal client.

3.3.3 Server a Server.
Se la maggior parte dei messaggi tra due server viene distribuita a tutti gli "altri" server, questo è in effetti richiesto solamente per ogni messaggio che abbia effetto su un utente, un canale o un server.

Dal momento che queste sono le voci base che si trovano in IRC, la quasi totalità dei messaggi originati da un server vengono distribuiti a tutti gli altri server connessi.

4. INFORMAZIONI DETTAGLIATE SUI SINGOLI MESSAGGI

Nelle pagine seguenti vengono descritti tutti i messaggi riconosciuti da un server e da un client IRC. Tutti i comandi descritti in questa sezione devono essere implementati su ogni server per questo protocollo.

Quando viene data la risposta ERR_NOSUCHSERVER, significa che non è stato possibile trovare il parametro <server>. Il server non deve dare ulteriori risposte per questo comando.

Il server, al quale un client è connesso, deve analizzare il messaggio completo, reinviando al mittente ogni eventuale errore.

Se il server si imbatte in un errore fatale durante l'analisi di un messaggio, un messaggio di errore deve essere inviato al mittente e l'analisi deve essere interrotta. Possono essere considerati errori fatali: un comando non corretto, una destinazione che risulta in qualche modo sconosciuta al server (server, nick, nome del canale rientrano in questa categoria), un numero insufficiente di parametri oppure un privilegio non corretto.

Se viene presentato un set completo di parametri, allora di ognuno di essi deve essere controllata la validità, ed eventuali risposte appropriate deve inviate al client. Nel caso di un messaggio che usi la virgola come separatore dei parametri, una risposta deve essere mandata per ogni voce.

Negli esempi sotto elencati, alcuni messaggi usano il formato completo previsto:

:Name COMMAND parameter list

Un tale esempio rappresenta un messaggio proveniente da "Name" in transito tra due servers, dove è essenziale includere il nome del mittente originario del messaggio, così che i server remoti possano mandare una risposta attraverso il percorso corretto.

4.1 Registrazione di Connessione.
I comandi qui descritti sono usati per registrare una connessione con un server IRC, sia come utente che come server, e per sconnettersi in maniera corretta.

Non è necessario un comando "PASS" perchè la connessione di un utente o di un server sia registrata, ma, nel caso ci fosse, esso deve precedere il messaggio del server oppure l'ultimo degli elementi della combinazione NICK/USER. è fortemente raccomandato che tutte le connessioni al server abbiano una password in modo da dare un certo livello di sicurezza alle connessioni attuali. L'ordine raccomandato dei comandi da inviare per la registrazione di un client è il seguente:

1. messaggio Pass
2. messaggio Nick

3. messaggio User

4.1.1 Messaggio Password
Comando: PASS
Parametri: <password>

Il comando PASS è usato per stabilire una parola d'ordine per la connessione. La parola d'ordine può e deve essere stabilita prima che sia fatto qualsiasi tentativo di registrare la connessione presso il server. Normalmente questo richiede che il client invii un comando PASS prima di inviare la combinazione NICK/USER ed il server *deve* inviare un comando PASS prima di ogni comando SERVER. La password fornita deve essere uguale a quella contenuta nelle righe C/N (per i servers) o nelle righe I (per i clients). è possibile inviare comandi PASS multipli prima di registrarsi, ma solamente l'ultimo comando inviato viene usato per la verifica della parola e non può essere cambiato una volta registrato.

Risposte numeriche:

ERR_NEEDMOREPARAMS (Errore_necessitano più parametri) ERR_ALREADYREGISTRED (Errore_già registrato)

Esempio:

PASS parola_d_ordine_qui

4.1.2 Messaggio Nickname
Comando: NICK
Parametri: <nickname> [<hopcount>]

Il messaggio NICK è usato per dare un nickname ad un utente o per cambiare quello che aveva in precedenza. Il parametro <hopcount> è usato solamente dai server per indicare quanto disti un nick dal suo server originario.

Una connessione locale ha un hopcount di 0. Se l'hopcount viene fornito da un client, deve essere ignorato.

Se un messaggio NICK arriva ad un server che è già a conoscenza di un identico nickname per un altro client, si verifica una collisione di nicknames. Il risultato di una collisione di nickname è la rimozione, dal database del server, di tutte le presenze del nickname, e un comando KILL viene emesso per rimuovere il nickname da tutti i database degli altri servers. Se il messaggio NICK, causa della collisione, era un cambio di nickname, allora anche il nick originale (quello vecchio) deve essere rimosso.

Se il server riceve un NICK già presente da un client che è connesso direttamente, può inviare al client locale un messaggio di ERR_NICKCOLLISION, ignorare il comando NICK, e non generare alcun kill.

Risposte numeriche:

ERR_NONICKNAMEGIVEN (Errore_non è stato dato nessun nickname)
ERR_ERRONEUSNICKNAME (Errore_nickname errato)
ERR_NICKNAMEINUSE (Errore_nickname già in uso)
ERR_NICKCOLLISION (Errore_collisone di nicknames)

Esempi:

NICK Wiz ; Inserimento nuovo nick "Wiz".

:WiZ NICK Kilroy ; WiZ cambia il suo nickname in Kilroy.

4.1.3 Messaggio User, utente
Comando: USER

Parametri: <nome_utente> <nome_macchina> <nome_server> <nome_reale>

Il messaggio USER è usato, all'inizio di una connessione, per specificare il nome dell'utente, della macchina dell'utente, del server ed il vero nome del nuovo utente. è usato, inoltre, nelle comunicazioni tra servers, per indicare un nuovo utente in arrivo su IRC dal momento che un utente è registrato solamente dopo che i comandi USER e NICK sono stati ricevuti da un client.

Tra i server il comando USER deve essere preceduto dal NICKname del client. Notare che il nome dell'host e del server sono normalmente ignorati dal server IRC quando il comando USER arriva da un utente connesso direttamente (per ragioni di sicurezza), ma sono usati nelle comunicazioni server-server. Questo significa che un comando NICK deve sempre essere inviato ad un server remoto quando un nuovo utente viene presentato al resto della rete, prima che il relativo comando USER sia inviato. Bisogna prestare attenzione al fatto che il parametro nome_reale deve essere l'ultimo, perchè può contenere dei caratteri spazio, e deve essere preceduto dal carattere due punti (':') per far sì che sia riconosciuto.

Dal momento che per l'utente è facile mentire sul suo nome-utente facendo affidamento unicamente sul messaggio USER, è raccomandato l'uso di un "Identity Server". Se la macchina dalla quale un utente si connette ha un server in grado di esplicare questa funzione, il nome_utente adottato sarà quello proveniente dalla risposta dell' "Identity Server".

Risposte numeriche:

ERR_NEEDMOREPARAMS (Errore_necessitano più parametri)
ERR_ALREADYREGISTRED (Errore_già registrato)

Esempi:

USER guest tolmoon tolsun :Ronnie Reagan ; Utente che registra se stesso con l'username "guest" ed il vero nome "Ronnie Reagan".

:testnick USER guest tolmoon tolsun :Ronnie Reagan ; messaggio tra server con il nickname al quale appartiene il comando USER.

4.1.4 Messaggio Server

Comando: SERVER
Parametri: <nome_server> <numero_tratte> <info>

Il messaggio server viene utilizzato per comunicare ad un server che dall'altra parte di una nuova connessione c'è un server.

Questo messaggio viene anche usato per passare i dati del server a tutta la rete. Quando un nuovo server è connesso alla rete, le informazioni che lo riguardano sono inviate a tutto il network. <numero-tratte> viene usato per dare a tutti i server alcune informazioni interne sulla distanza cui si trova ciascun server. Con una lista completa dei server, sarebbe possibile ricostruire una mappa completa della configurazione dei collegamenti fra i server, ma l'hostmask ne impedisce la realizzazione.

Il messaggio SERVER deve essere accettato solamente: (a) o da una connessione non ancora registrata e sta cercando di registrarsi come server; (b) o da una connessione già esistente con un altro server, nel qual caso il messaggio SERVER introduce un nuovo server che si trova dietro quel server.

La maggior parte degli errori che si verificano con la ricezione di un comando SERVER consistono in una interruzione della connessione da parte del server di destinazione (target SERVER).

Di solito le risposte di errore sono inviate usando il comando ERROR, piuttosto che inviare quelle numeriche, dal momento che il comando ERROR ha diverse proprietà utili che lo rendono, in questo caso, vantaggioso. Se un messaggio SERVER viene analizzato e cerca di introdurre un server che è già noto al server ricevente, la connessione dalla quale arriva il messaggio deve essere chiusa (se si seguono le procedure corrette), in quanto si è formato un doppio percorso per quel server e la natura aciclica della rete IRC risulta compromessa.

Risposta Numerica:

ERR_ALREADYREGISTRED (Errore_già registrato)

Esempi:

SERVER test.oulu.fi 1 :[tolsun.oulu.fi] Experimental server ; Il nuovo server test.oulu.fi si introduce e cerca di registrarsi. Il nome tra parentesi [..] è il nome della macchina sulla quale è attivo test.oulu.fi.

:tolsun.oulu.fi SERVER csd.bu.edu 5 :BU Central Server ; Il server tolsun.oulu.fi è il nostro collegamento per csd.bu.edu che si trova a 5 tratte di distanza.

4.1.5 Messaggio Oper, Operatore

Comando: OPER
Parametri: <utente> <parola_d_ordine>

Il messaggio OPER è usato da un utente normale per ottenere i privilegi di operatore. La combinazione di <utente> e <parola_d_ordine> è richiesta per avere quei privilegi. Se il client che invia il comando OPER fornisce la corretta parola d'ordine per l'utente dato, il server informa del nuovo operatore il resto della rete effettuando un "MODE +o" per il nickname del client. Il messaggio OPER è solamente un messaggio client-server.

Risposte numeriche:

ERR_NEEDMOREPARAMS (Errore_necessitano più parametri)
RPL_YOUREOPER (risposta_sei già operatore)
ERR_NOOPERHOST (errore_non ci sono O: line per l'utente specificato)
ERR_PASSWDMISMATCH (errore_la password non corrisponde)

Esempio:

OPER foo bar ; Tentativo di registrare come un operatore usando "foo" come nome utente e "bar" come parola d'ordine.

4.1.6 Messaggio Quit (abbandonare)

Comando: QUIT
Parametri: [<Messaggio di cessazione>]

La sessione di un client è terminata con un messaggio di cessazione. Il server deve chiudere la connessione con un client che invia un messaggio QUIT. Se viene dato un "Messaggio di cessazione", verrà mostrato questo al posto del messaggio di default, consistente nel semplice nickname.

Quando un tratto della rete si interrompe (sconnessione di due server) "split", il messaggio quit si compone dei nomi dei due server coinvolti separati da uno spazio. Il primo nome è quello del server che è ancora connesso, il secondo quello del server sconnesso. Se, per qualche altra ragione, la connessione con un client viene chiusa senza che il client abbia inviato un messaggio QUIT (per esempio il client si pianta e si verifica un EOF (End Of File) sul socket) (4), il server deve compilare il messaggio di abbandono con una formulazione che rifletta la natura dell'evento che ha causato la sconnessione.

Risposte numeriche:

Nessuna.

Esempio:

QUIT :Gone to have lunch (Andato a pranzo) ; Format preferibile del messaggio.

4.1.7 Messaggio Server Quit
Comando: SQUIT
Parametri: <server> <commento>

Il messaggio SQUIT viene impiegato per rendere conto dei server che abbandonano oppure che smettono di funzionare. Se un server desidera interrompere la connessione con un altro server deve mandare un messaggio SQUIT all'altro server usando il nome dell'altro server, che chiude la sua connessione col server che abbandona, come parametro "server".

La disponibilità di questo comando agli operatori anche per mantenere ordinato, l'assetto delle connessioni all'interno della rete IRC. Gli operatori possono dunque usare il messaggio SQUIT per una connessione ad un server remoto. In questo caso lo SQUIT deve essere analizzato da ogni server tra l'operatore ed il server remoto, aggiornando la "visione" della rete che ciascun server deve conservare, come viene spiegato più sotto. Il parametro "commento" dovrebbe essere fornito da tutti gli operatori che eseguono uno SQUIT su un server remoto (che non è connesso cioè al server sul quale essi operano) cosicchè gli altri operatori siano informati sui motivi del provvedimento. Il "commento" viene definito anche dai server, che possono formularlo come un messaggio di errore o simili.

Ambedue i server che si trovino agli estremi della connessione che viene chiusa devono inviare un messaggio SQUIT (a tutte le altre connessioni server che li riguardano) per tutti gli altri server che sono considerati trovarsi a valle di ciascuno di questi collegamenti. Alla stessa maniera, un messaggio QUIT deve essere inviato a tutti gli altri server connessi al resto della rete nell'interesse di tutti i client che si trovino a valle di quel collegamento. In aggiunta a tutto questo, a tutti i membri di un canale, i quali perdono un membro a causa dello split, deve essere inviato un messaggio QUIT.

Se la connessione ad un server viene terminata prematuramente (per esempio il server dalla parte opposta del link smette di funzionare), al server che rileva questa sconnessione è richiesto di informare il resto della rete che la connessione si è chiusa e di compilare il campo "commento" con qualcosa di appropriato.

Risposte numeriche:

ERR_NOPRIVILEGES (errore_non hai i privilegi dell'operatore)
ERR_NOSUCHSERVER (errore_ il serve indicato non esiste)

Esempi:

SQUIT tolsun.oulu.fi :Bad Link ; la connessione al server tolson.oulu.fi è stata terminata a causa di un "Bad Link" (cattiva connessione).

:Trillian SQUIT cm22.eng.umd.edu :Server out of control ; messaggio da Trillian per disconnettere "cm22.eng.umd.edu" dalla rete perchè "Server out of control" (server fuori controllo).

4.2 Operazioni del Canale
Questo gruppo di messaggi concerne la manipolazione dei canali, le loro proprietà (mode del canale), ed il loro contenuto (usualmente clients).
Per una giusta realizzazione, è necessario stabilire un numero di regole, quando i client che si trovano agli estremi opposti della rete inviano messaggi che potrebbero collidere fra loro. è richiesto, inoltre, che i server mantengano una traccia storica dei nickname per assicurarsi che, ogniqualvolta un parametro <nick> è dato, il server ne controlli la storia nel caso sia stato cambiato di recente.

4.2.1 Messaggio Join
Comando: JOIN
Parametri: <canale>{,<canale>} [<chiave>{,<chiave>}]

Il comando JOIN è usato dal client per avere accesso ai messaggi provenienti da un dato canale. La verifica della sussistenza della condizione perchè un client sia autorizzato o meno ad aggregarsi al canale, viene realizzata dal server al quale il client è connesso. Tutti gli altri server aggiungono automaticamente il client al

canale quando la sua ammissione ad un canale è notificata da altri server. Le condizioni che determinano l'ammissione sono le seguenti:

1. Il client deve essere invitato se il canale è ad invito (mode +i);
2. Il nick/nome_utente/nome_macchina del client non devono corrispondere a ban attivi;
3. Se è definita una parola d'ordine di acceso, deve essere fornita la parola corretta (mode +k) (5).

Tutto questo è discusso più dettagliatamente nella sezione Comando MODE (si veda la sezione 4.2.3 per maggiori informazioni).

Una volta che gli utenti si sono aggregati al canale, essi ricevono notizia di tutti i comandi, ricevuti dal loro server, che riguardano il canale. Questo Include MODE, KICK, PART, QUIT e naturalmente PRIVMSG/NOTICE. Il comando JOIN necessita di essere trasmesso a tutti i server di modo che ogni server sappia dove trovare gli utenti che sono sul canale. Questo permette una ottimizzazione nell'invio dei messaggi di tipo PRIVMSG/NOTICE al canale. Se un JOIN ha successo, all'utente vengono notificati l' "argomento" del canale (topic) - usando RPL_TOPIC - e la lista degli utenti che sono nel canale, che include l'utente aggregato, (usando RPL_NAMREPLY).

Risposte numeriche:

ERR_NEEDMOREPARAMS (errore_necessitano più parametri)
ERR_BANNEDFROMCHAN (errore_bannato dal canale)
ERR_INVITEONLYCHAN (errore_canale ad invito)
ERR_BADCHANNELKEY (errore_key sbagliata)
ERR_CHANNELISFULL (errore_canale pieno)
ERR_BADCHANMASK (errore_Chanmask errata)
ERR_NOSUCHCHANNEL (errore_canale inesistente)
ERR_TOOMANYCHANNELS (errore_troppi canali)
RPL_TOPIC (risposta_topic del canale:) (6)

Esempi:

JOIN #foobar ; richiesta di aggregazione al canale #foobar.

JOIN &foo fubar ; richiesta di aggregazione al canale &foo usando la parola d'ordine "fubar".

JOIN #foo,&bar fubar ; richiesta di aggregazione al canale #foo usando la parola d'ordine "fubar" ed al canale &bar usando nessuna key.

JOIN #foo,#bar fubar,foobar ; richiesta di aggregazione al canale #foo usando la parola d'ordine "fubar" ed al canale #bar usando "foobar".

JOIN #foo,#bar ; richiesta di aggregazione ai canali #foo e #bar.

:WiZ JOIN #Twilight_zone ; messaggio JOIN da WiZ

4.2.2 Messaggio Part
Comando: PART
Parametri: <canale>{,<canale>}

Il messaggio PART comporta la rimozione del client mittente del messaggio dalla lista di utenti attivi di tutti i canali elencati nella riga dei parametri.

Risposte numeriche:

ERR_NEEDMOREPARAMS (errore_necessitano più parametri)
ERR_NOSUCHCHANNEL (errore_canale inesistente)
ERR_NOTONCHANNEL (errore_non sei sul canale)

Esempi:

PART #twilight_zone ; richiesta di lasciare il canale "#twilight_zone"

PART #oz-ops,&group5 ; richiesta di lasciare sia il canale "&group5" che "#oz-ops".

4.2.3 Messaggio Mode

Comando: MODE

Il comando MODE ha due scopi: permette infatti di cambiare i MODE sia dei canali che degli utenti. La ragione di questa scelta è che in futuro i nickname saranno obsoleti e l'entità equivalente sarà il canale. (7)

Quando viene analizzato un comando MODE, è essenziale che prima il messaggio venga analizzato nella sua totalità e che, in un secondo tempo, i cambiamenti dettati dal messaggio vengano effettuati.

4.2.3.1 Mode dei canali

Parametri: <canale> {[+|-]|o|p|s|i|t|n|b|v} [<limite>] [<utente>] [<schema_di_ban>] (8) Il comando MODE è a disposizione degli operatori di canale perchè possano cambiare le caratteristiche del 'loro' canale. è richiesto inoltre che i server siano in grado di cambiare i mode dei canali così che sia possibile creare operatori. I mode dei canali disponibili sono i seguenti:

o - dà/toglie i privilegi di operatore;
p - permette di definire il canale come privato;
s - permette di definire il canale come segreto;
i - permette di definire il canale come canale ad invito;
t - permette di definire il topic come definibile solo da operatori del canale;
n - permette di stabilire la proibizione di invio di messaggi da client che sono al di fuori del canale;
m - permette di definire il canale come canale moderato;
l - stabilisce un limite al numero di utenti presenti sul canale;
b - definisce uno schema di ban per tenere gli utenti fuori del canale;
v - dà/toglie la possibilità di parlare su un canale moderato
k - setta una parola d'ordine per l'acceso al canale.

Quando si usano le opzioni 'o' e 'b', è stata imposta una restrizione di tre comandi per mode. Vale a dire, ogni combinazione di 'o' e (Qui il testo inglese è lacunoso, ma, con ogni probabilità si intende dire che ogni combinazione di 'o' e 'b', può contenere al massimo tre elementi. - N.d.T.)

4.2.3.2 Mode dell'utente

Parametri: <nickname> {[+|-]|i|w|s|o} (9)

I MODE degli utenti sono cambiamenti che determinano come il client è visto dagli altri, oppure che tipo di messaggi 'extra' possono essere inviati al client. Il comando MODE per l'utente può essere accettato solamente se il mittente del messaggio e il nickname dato come parametro sono identici.

I mode disponibili sono i seguenti:

i - definisce un utente come invisibile;
s - determina la ricezione da parte dell'utente delle notizie del server;
w - permette all'utente di ricevere i wallops (messaggi tra operatori);
o - permette di definire l' utente come operatore.

Mode in aggiunta a questi potranno essere disponibili prossimamente. Se un utente cerca di farsi operatore da solo, usando il mode "+o", il tentativo è ignorato. Non c'è alcuna restrizione, comunque, per chiunque voglia deopparsi (usando "-o").

Risposte numeriche:

ERR_NEEDMOREPARAMS (errore_necessitano più parametri)
RPL_CHANNELMODEIS (risposta_il modo di canale è:)
ERR_CHANOPRIVSNEEDED (errore_sono richiesti le prerogative di operatore di canale)
ERR_NOSUCHNICK (errore_nick inesistente)
ERR_NOTONCHANNEL (errore_non sei sul canale)
ERR_KEYSET (errore_nella definizione della parola d'ordine)
RPL_BANLIST (risposta_lista dei ban:)
RPL_ENDOFBANLIST (risposta_fine della lista dei ban:)
ERR_UNKNOWNMODE (errore_modo sconosciuto)
ERR_NOSUCHCHANNEL (errore_canale non esistente)
ERR_USERSDONTMATCH (errore_user non corrispondente)
RPL_UMODEIS (risposta_il mode dell'utente è:)
ERR_UMODEUNKNOWNFLAG (errore_flag sconosciuto per un mode utente)

Esempi:

Uso dei MODE del canale:

MODE #Finnish +im ; rende il canale #Finnish moderato e ad invito.

MODE #Finnish +o Kilroy ; da i privilegi di operatore di canale a Kilroy sul canale #Finnish.

MODE #Finnish +v Wiz ; permette a WiZ di parlare in #Finnish.

MODE #Fins -s ; rende il canale #Fins non più segreto.

MODE #42 +k oulu ; definisce "oulu" come parola d'ordine per l'accesso al canale.

MODE #eu-ofors +l 10 ; setta a 10 il numero limite di utenti presenti sul canale.

MODE &oulu +b ; lista gli schemi di ban definiti per il canale.

MODE &oulu +b *!*@* ; impedisce l'accesso al canale di tutti gli utenti.

MODE &oulu +b *!*@*.edu ; impedisce l'accesso al canale di ogni utente il cui nome di macchina corrisponda ad *.edu.

Uso dei MODE dell'utente:

:MODE WiZ -w ; disabilita WIZ dalla ricezione dei wallops.

:Angel MODE Angel +i ; messaggio da Angel di rendersi invisibile.

MODE WiZ -o ; WiZ si deoppa (rimuovendo lo status di operatore). Il contrario ovvio di questo comando non deve essere permesso agli utenti dal momento che aggirerebbe il comando OPER.

4.2.4 Messaggio Topic

Comando: TOPIC
Parametri: <canale> [<topic>]

Il messaggio TOPIC è usato per cambiare o prender visione del topic di un canale. Se non è dato alcun parametro <topic> il messaggio di ritorno sarà il topic del canale. Se invece il parametro <topic> è presente, il topic di quel canale potrà essere cambiato, se il mode del canale permette questa azione.

Risposte numeriche:

ERR_NEEDMOREPARAMS (errore_necessitano più parametri)
ERR_NOTONCHANNEL (errore_non sei sul canale)
RPL_NOTOPIC (risposta_topic vuoto)

RPL_TOPIC (risposta_il topic del canale è:)
ERR_CHANOPRIVSNEEDED (errore_necessitano i privilegi dell'operatore)

Esempi:

:Wiz TOPIC #test :New topic ;l'utente Wiz definisce il topic.

TOPIC #test :another topic ;definisce il topic su #test come "another topic".

TOPIC #test ; controlla il topic del canale #test.

4.2.5 Messaggio Names

Comando: NAMES
Parametri: [<canale>{,< canale>}]

Usando il comando NAMES gli utenti possono avere una lista di tutti i nickname che siano loro visibili, su ogni canale che essi vedono. I nomi dei canali che possono vedere sono quelli che non sono privati (+p) o segreti (+s) o quelli sui quali si trovano. Il parametro <canale> specifica a proposito di quale/i canale/i inviare l'informazione richiesta, se questa risulta valida. Non c'è risposta di errore in caso di nome errato del canale.

Se non è dato alcun parametro <canale>, viene fornita una lista di tutti i canali e dei loro occupanti. Alla fine della lista, viene fornito un elenco degli utenti (come appartenenti ad un canale "*") che sono visibili, ma non su ogni canale o invisibili su un canale visibile.

Risposte numeriche:

RPL_NAMREPLY (risposta_risposta dei nomi:)
RPL_ENDOFNAMES (risposta_fine dei nomi:)

Esempi:

NAMES #twilight_zone,#42 ; lista gli utenti visibili sui canali #twilight_zone e #42 se i canali sono visibili.

NAMES ; elenca tutti i canali ed utenti visibili.

4.2.6 Messaggio List

Comando: LIST Parametri: [<canale>{,<canale>} [<server>]]

Il messaggio LIST è usato per listare i canali ed i loro topics. Se è presente il parametro <canale>, viene mostrato solo lo status di quel canale. I canali privati vengono listati (senza il loro topic) come "Prv" a meno che il client da cui proviene la domanda di informazione, non sia su quel canale. Alla stessa maniera, i canali segreti non vengono elencati, a meno che il client non sia un membro del canale in questione.

Risposte numeriche:

ERR_NOSUCHSERVER (errore_server inesistente)
RPL_LISTSTART (risposta_inizio della lista:)
RPL_LIST (risposta_lista:)
RPL_LISTEND (risposta_fine della lista:)

Esempi:

LIST ; lista tutti i canali.

LIST #twilight_zone,#42 ; lista i canali #twilight_zone e #42

4.2.7 Messaggio Invite

Comando: INVITE Parametri: <nickname> <canale>

Il messaggio INVITE usato per invitare gli utenti su un canale. Il parametro <nickname> è il nick della persona da invitare sul canale <canale>. Non è necessario che il canale in cui <nickname> viene invitato, debba esistere o debba essere un canale valido. Per invitare un utente in un canale ad invito (MODE +i) si deve essere riconosciuti come operatori del canale.

Risposte numeriche:

ERR_NEEDMOREPARAMS (errore_necessitano più parametri)
ERR_NOSUCHNICK (errore_nick inesistente)
ERR_NOTONCHANNEL (errore_non sei sul canale)
ERR_USERONCHANNEL (errore_utente già sul canale)
ERR_CHANOPRIVSNEEDED (errore_necessitano i privilegi dell'operatore)
RPL_INVITING (risposta_invito a:)
RPL_AWAY (rispoasta_utente in away:)

Esempi:

:Angel INVITE Wiz #Dust ; l'utente Angel invita WiZ sul canale #Dust

INVITE Wiz #Twilight_Zone ; comando di invito di WiZ su #Twilight_zone

4.2.8 Messaggio Kick

Comando: KICK
Parametri: <canale> <utente> [<commento>]

Il comando KICK può essere usato per rimuovere forzatamente un utente da un canale. Esso lo 'scalcia fuori del canale' (PART forzato). Solo un operatore di canale può rimuovere un utente dal canale. Ogni server che riceva un messaggio KICK controlla se il comando sia valido (per esempio se il mittente è un operatore), prima di rimuovere la vittima dal canale.

Risposte numeriche:

ERR_NEEDMOREPARAMS (errore_necessitano più parametri)
ERR_NOSUCHCHANNEL (errore_nick inesistente)
ERR_BADCHANMASK (errore_schema di nome di canale errato)
ERR_CHANOPRIVSNEEDED (errore_necessitano i privilegi dell'operatore)
ERR_NOTONCHANNEL (errore_non sei sul canale)

Esempi:

KICK &Melbourne Matthew ; caccia Matthew da &Melbourne

KICK #Finnish John :Speaking English ; caccia John from #Finnish giustificando l'azione con "Speaking English" (commento).

:WiZ KICK #Finnish John ; messaggio di KICK da WiZ per rimuovere John dal canale #Finnish

Nota: è possibile estendere i parametri del comando KICK come segue:

<canale>{,<canale>} <utente>{,<utente>} [<commento>] (10)

4.3 Comandi e richieste di informazioni al server

Il gruppo di comandi per la richiesta di informazioni al server è stato previsto per ottenere informazioni su ogni server connesso alla rete. Tutti i server connessi devono replicare alle richieste di informazioni in maniera corretta. Ogni risposta non corretta (o qualunque mancanza di risposta) deve essere considerata come un mal funzionamento da parte del server che deve essere sconnesso e/o disabilitato prima possibile e

finchè non si sia posto rimedio alla disfunzione. In quelle domande, dove compare un parametro "<server>", generalmente significa che il parametro potrebbe essere un nickname o un server oppure un nome jolly di qualche sorta. Per ogni parametro, comunque, vengono generati solamente una domanda ed un set di risposte.

4.3.1 Messaggio Version

Comando: VERSION
Parametri: [<server>]

Il messaggio VERSION è usato per avere informazioni a proposito della versione nel programma implementato sul server. Un parametro facoltativo <server> viene usato per informarsi sulla versione del programma attivo su un server al quale un client non sia connesso direttamente.

Risposte numeriche:

ERR_NOSUCHSERVER (errore_server inesistente)
RPL_VERSION (risposta_La versione è:)

Esempi:

:Wiz VERSION *.se ; messaggio da Wiz per informarsi sulla versione di un server il cui nome corrisponda allo schema: "*.se"

VERSION tolsun.oulu.fi ; richiesta di informazione sulla versione del server "tolsun.oulu.fi".

4.3.2 Messaggio Stats

Comando: STATS
Parametri: [<richiesta> [<server>]]

Il messaggio STATS usato per chiedere informazioni statistiche su un determinato server. Se il parametro <server> viene omesso, viene inviata solamente la risposta di fine del messaggio stats. Le modalità di esecuzione di questo comando dipendono fortemente dal server che risponde, tuttavia i server debbono essere in grado di fornire le informazioni descritte qui di seguito (o qualcosa di analogo). Una particolare richiesta può essere denotata da una singola lettera controllata dal solo server di destinazione (se dato come parametro <server>), ed è invece trasmessa, ignorata ed inalterata, dai server intermedi. Le richieste di informazioni che seguono, sono quelle reperibili nell'attuale implementazione di IRC, e costituiscono una porzione abbondante delle informazioni di setup per il server in questione. Malgrado queste richieste possano essere soddisfatte con modalità diverse da altre versioni, tutti i server dovrebbero essere in grado di fornire una risposta valida ad una richiesta STAT, una replica cioè, conforme ai formati di risposta comunemente usati e coerente con lo scopo della richiesta.

Le richieste usualmente soddisfatte sono:

c - ritorna una lista di server ai quali il server può connettersi o dai quali permette connessioni;
h - ritorna una lista di server i quali sono considerati forzatamente "foglie" (elementi periferici) nell'albero della rete oppure ai quali è permesso di agire come suoi punti nodali;
i - ritorna una lista di host provenendo dai quali il server permette ad un client di connettersi;
k - ritorna una lista di combinazioni nome_utente/nome_macchina bannati su quel server;
l - ritorna una lista delle connessioni del server, mostrando da quanto tempo si è stabilita ogni connessione, il traffico su quella connessione in byte e messaggi per ogni direzione;
m - ritorna una lista di comandi riconosciuti dal server ed il conteggio del numero di volte in cui ogni comando è stato usato, se il conteggio è un numero diverso da zero;
o - ritorna una lista di host provenendo dai quali normali client sono autorizzati a diventare operatori;
y - mostra Y (Class) linee del file di configurazione del server;
u - ritorna una riga che mostra da quanto tempo il server è attivo. (11)

Risposte numeriche:

ERR_NOSUCHSERVER (errore_server inesistente)
RPL_STATSCLINE   (risposta_ad una richiesta di tipo "c")
RPL_STATSHLINE   (risposta_ad una richiesta di tipo "h")
RPL_STATSILINE   (risposta_ad una richiesta di tipo "i")
RPL_STATSKLINE   (risposta_ad una richiesta di tipo "k")
RPL_STATSLLINE   (risposta_ad una richiesta di tipo "l")
RPL_STATSNLINE   (risposta_ad una richiesta di tipo "n")
RPL_STATSOLINE   (risposta_ad una richiesta di tipo "o")
RPL_STATSQLINE   (risposta_ad una richiesta di tipo "q")
RPL_STATSLINKINFO (risposta_ad una richiesta di informazioni sullo stato delle connessioni)
RPL_STATSUPTIME   (risposta_ad una richiesta di tipo "u")
RPL_STATSCOMMANDS (risposta_ad una richiesta di tipo "m")
RPL_ENDOFSTATS   (risposta_fine delle informazioni statistiche)

Esempi:

STATS m ; controlla le statistiche di utilizzo dei comandi sul server al quale si è connessi

:Wiz STATS c eff.org ; richiesta di WiZ per informazioni sulle linee C/N al server eff.org

4.3.3 Messaggio Links

Comando: LINKS
Parametri: [[<server remoto>] <schema_server>]

Col comando LINKS, un utente può ottenere l'elenco di tutti i server conosciuti al server che risponde alla richiesta di informazione. La lista di server inviata deve essere compatibile con lo schema di nome del server fornito nel secondo parametro, se non viene data alcuno schema, viene inviata la lista completa. Se il parametro <server remoto> viene dato in aggiunta a <schema_server>, il comando LINKS viene inoltrato al primo server il cui nome (sempre se esista) corrisponde a quello contenuto in <server remoto>, e a quel server è allora richiesto di rispondere alla domanda.

Risposte numeriche:

ERR_NOSUCHSERVER (errore_server inesistente)
RPL_LINKS   (risposta_lista collegamenti)
RPL_ENDOFLINKS   (risposta_fine delle lista collegamenti)

Esempi:

LINKS *.au ; lista tutti i server con un nome coerente con lo schema_server *.au;

:WiZ LINKS *.bu.edu *.edu ; messaggio LINKS da WiZ al primo server il cui nome è coerente con lo schema *.edu per una lista di server che soddisfano lo schema *.bu.edu.

4.3.4 Messaggio Time
Comando: TIME
Parametri: [<server>]

Il messaggio TIME viene usato per chiedere l'ora locale relativa al server specificato. Se non viene dato un parametro server, il server che tratta il comando deve rispondere alla domanda.

Risposte numeriche:

ERR_NOSUCHSERVER (errore_server inesistente)
RPL_TIME   (risposta_ora locale)

Esempi:

TIME tolsun.oulu.fi ; controlla l'orario sul server "tolson.oulu.fi"

Angel TIME *.au ; l'utente Angel controlla l'orario su un server corrispondente allo schema "*.au"

4.3.5 Messaggio Connect

Comando: CONNECT
Parametri: <server_destinazione> [<porta> [<server remoto>]]

Il comando CONNECT può essere usato per forzare un server a stabilire una nuova ed immediata connessione ad un altro server. CONNECT è un comando privilegiato ed è disponibile solamente agli operatori IRC. Dato un server remoto, allora il tentativo di connessione viene effettuato da quel server al <erver_destinazione> attraverso la porta <porta>. [Ndr - Pur essendo il termine inglese originale: "port" equivalente all'italiano "porto", si mantiene qui l'erronea traduzione "port" entrata ormai nella tradizione tecnico-informatica dell' italiano]

Risposte numeriche:

ERR_NOSUCHSERVER (errore_server inesistente)
ERR_NOPRIVILEGES (errore_non hai i privilegi dell'operatore)
ERR_NEEDMOREPARAMS (errore_necessitano più parametri)

Esempi:

CONNECT tolsun.oulu.fi ;tentativo di connettere un server a tolsun.oulu.fi

:WiZ CONNECT eff.org 6667 csd.bu.edu ;tentativo effettuato da WiZ di far connettere i server eff.org e csd.bu.edu sulla porta 6667.

4.3.6 Messaggio Trace

Comando: TRACE
Parametri: [<server>]

Il comando TRACE è impiegato per trovare il percorso verso un server specificato. Ogni server che riceve e tratta questo messaggio, deve comunicare al server mittente informazioni su se stesso, inviando una risposta e denotandosi come un collegamento di passaggio. Si forma così una catena di risposte, simile a quella che si riceve usando il "traceroute". Dopo aver inviato la risposta, il server deve mandare il messaggio TRACE al server successivo, finchè il server il cui nome è contenuto nel parametro <server> non viene raggiunto. Se viene omesso il parametro <server>, è previsto che il comando TRACE invii un messaggio al mittente, indicando con quali server il server corrente ha una connessione diretta. Se la destinazione data da <server> è un server effettivamente collegato, allora al server di destinazione è è fatto obbligo di notificare tutti i server e gli utenti connessi, anche se, in effetti, solo agli operatori è permesso vedere gli utenti presenti. Se la destinazione data da <server> è il nick di un utente, allora solo una risposta per quel nick viene data. (12)

Risposte numeriche:

ERR_NOSUCHSERVER (errore_server inesistente)

Se il messaggio TRACE è indirizzato ad un altro server, tutti i server intermedi devono inviare una risposta RPL_TRACELINK per indicare che il messaggio TRACE è passato attraverso di loro e specificare quale sarà la destinazione immediatamente successiva.

RPL_TRACELINK (risposta_traccia di collegamento)

Una risposta TRACE può essere composta da un numero qualsiasi delle seguenti risposte numeriche.

RPL_TRACECONNECTING (risposta_TRACE: collegamento)
RPL_TRACEHESHAKE (risposta_TRACE: riconoscimento (handshaking))
RPL_TRACEUNKNOWN (risposta_TRACE: elemento sconosciuto)

RPL_TRACEOPERATOR (risposta_TRACE: operatore)
RPL_TRACEUSER (risposta_TRACE: utente)
RPL_TRACESERVER (risposta_TRACE: server)
RPL_TRACESERVICE (risposta_TRACE: servizio)
RPL_TRACENEWTYPE (risposta_TRACE: nuovo tipo)
RPL_TRACECLASS (risposta_TRACE: classe)

Esempi:

TRACE *.oulu.fi ;TRACE ad un server il cui nome è coerente con lo schema *.oulu.fi

:WiZ TRACE AngelDust ;TRACE emesso da WiZ per il nick AngelDust

4.3.7 Messaggio Admin

Comando: ADMIN
Parametri: [<server>]

Il messaggio ADMIN viene usato per trovare il nome dell'amministratore di un dato server, oppure del server corrente se il parametro <server> viene omesso. Ogni server deve avere la possibilità di inoltrare messaggi ADMIN ad altri server.

Risposte numeriche:

ERR_NOSUCHSERVER (errore_server inesistente)
RPL_ADMINME (risposta_ADMIN: me)
RPL_ADMINLOC1 (risposta_ADMIN: locale1)
RPL_ADMINLOC2 (risposta_ADMIN: locale2)
RPL_ADMINEMAIL (risposta_indirizzo e-mail dell'amministratore:)

Esempi:

ADMIN tolsun.oulu.fi ; richiesta di informazione ADMIN a tolsun.oulu.fi

:WiZ ADMIN *.edu ; ADMIN richiesta da WiZ per il primo server il cui nome è coerente con lo schema *.edu.

4.3.8 Messaggio Info

Comando: INFO
Parametri: [<server>]

Il comando INFO provoca l'invio di una risposta che descrive il server: la versione, data di compilazione, la release dell'ultimo aggiornamento applicato (patchlevel), quando è stato attivato, ed ogni altra informazione che può essere considerata rilevante.

Risposte numeriche:

ERR_NOSUCHSERVER (errore_server inesistente)
RPL_INFO (risposta_informazioni:)
RPL_ENDOFINFO (risposta_fine delle informazioni)

Esempi:

INFO csd.bu.edu ; richiesta di risposta INFO da csd.bu.edu

:Avalon INFO *.fi ; richiesta INFO da Avalon per il primo server il cuoi nome è coerente con lo schema *.fi.

INFO Angel ; richiesta INFO dal server al quale Angel è connesso.

4.4 Invio messaggi

Lo scopo principale del protocollo IRC è di fornire una base comune per tutti i client sulla quale essi possano comunicare gli uni con gli altri. PRIVMSG e NOTICE sono gli unici comandi a disposizione che di fatto inviano il testo di un messaggio da un client all'altro - tutto il resto esiste per rendere possibile questa azione e per assicurare che l'invio avvenga in modo strutturato ed affidabile.

4.4.1 Messaggi Privati

Comando: PRIVMSG
Parametri: <destinatario>{,<destinatario>} <testo da inviare>

PRIVMSG viene usato per inviare messaggi privati tra utenti. <destinatario> è il nickname del destinatario del messaggio. <destinatario> può anche essere una lista di nomi o canali, separata da virgole. Il parametro <destinatario> può anche essere uno schema di nome-macchina (#mask) oppure uno schema di nome-server ($mask). In ambedue i casi il server invierà il PRIVMSG a coloro che hanno un server oppure una macchina il cui nome è coerente con lo schema fornito. Lo schema deve contenere almeno un (1) punto "." e nessun carattere-jolly dopo l'ultimo punto ".". Questo è necessario per prevenire qualcuno dall'inviare messaggi del tipo "#*" o a "$*", che sarebbero trasmessi a tutti gli utenti. L'esperienza insegna che di questo comando viene fatto si abusa spesso in maniera irresponsabile. Per carattere-jolly si intendono i caratteri '*' e '?'.

Questa estensione al comando PRIVMSG è disponibile solo per gli operatori.

Risposte numeriche:

ERR_NORECIPIENT (errore_destinatario non specificato)
ERR_NOTEXTTOSEND (errore_testo non specificato)
ERR_CANNOTSENDTOCHAN (errore_invio impossibile al canale)
ERR_NOTOPLEVEL (errore_non livello massimo)
ERR_WILDTOPLEVEL (errore_livello massimo non giustificato)
ERR_TOOMANYTARGETS (errore_troppe destinazioni)
ERR_NOSUCHNICK (errore_nick inesistente)
RPL_AWAY (risposta_destinatario in away)

Esempi:

:Angel PRIVMSG Wiz :Hello are you receiving this message ? ; messaggio da Angel a Wiz.

PRIVMSG Angel :yes ìm receiving it !receiving it !'u>(768u+1n) .br ; messaggio a Angel.

PRIVMSG jto@tolsun.oulu.fi :Hello ! ; messaggio ad un client su un server tolsun.oulu.fi con nome-utente "jto".

PRIVMSG $*.fi :Server tolsun.oulu.fi rebooting. ; messaggio a chiunque sia su un server con un nome coerente con *.fi.

PRIVMSG #*.edu :NSFNet è undergoing work, expect interruptions ; messaggio a tutti gli utenti che hanno una macchina il cui nome è coerente con *.edu.

4.4.2 Messaggi Notice

Comando: NOTICE
Parametri: <nickname> <text>

Il messaggio NOTICE viene usato in modo simile al PRIVMSG. La differenza tra NOTICE e PRIVMSG è che una risposta automatica non deve mai essere inviata come replica ad un messaggio NOTICE. Questa regola si applica pure ai server - essi non devono ritornare una risposta di errore al client, alla ricezione di un NOTICE. Questa regola è utile per impedire l'innescarsi di cicli senza uscita tra un client che invia automaticamente qualcosa in risposta a qualcosa che ha ricevuto. Di solito si adotta questo metodo con gli automi (client che possiedono un modulo di Intelligenza artificiale oppure un qualsiasi altro programma

interattivo che controlla le loro azioni), i quali inviano immancabilmente delle risposte, per paura che finiscano in cicli senza fine con altri automi. [Ndr - Si tratta dei così detti bot - abbreviazione della parola ceca "robot", programmi implementati sulla macchina di un client che rimangono in linea ed hanno un comportamento del tutto automatico]

Si veda PRIVMSG per maggiori dettagli sulle risposte e gli Esempi.

4.5 Richieste di informazioni sull'utente

Le richieste di informazioni sull'utente sono un gruppo di comandi che concernono primariamente la ricerca di dettagli su un utente particolare oppure su un gruppo di utenti. Quando si usano dei caratteri-jolly per denotare l'utente in questi comandi, di tutti gli utenti che siano eventualmente coerenti con lo schema fornito, verranno inviate informazioni solo sugli utenti "visibili" al richiedente. La visibilità di un utente è determinata come una combinazione del mode dell'utente ed i comuni canali sui quali si trovano sia l'utente inquisito che il richiedente.

4.5.1 Richiesta Who

Comando: WHO
Parametri: [<nome> [<o>]]

Il messaggio WHO è usato da un client per generare una richiesta la cui risposta è una lista di informazioni su utenti che sono coerenti con il parametro <nome> fornito dal client. In assenza del parametro <nome>, tutti gli utenti visibili (utenti che non sono invisibili (modo-utente +i) e che non hanno un canale in comune con il client richiedente) vengono listati.

Lo stesso risultato può essere raggiunto usando un <nome> "0" o qualsiasi schema con caratteri-jolly con cui sarebbe coerente ogni possibile utente. Il <nome> trasmesso a WHO viene confrontato col nome della macchina dell'utente, il server, il "nome reale" ed il nick, se il canale <nome> non può essere trovato. Se il parametro "o" viene trasmesso, solo gli operatori, in funzione dello schema di nome-utente fornito, verranno listati.

Risposte numeriche:

ERR_NOSUCHSERVER (errore_server inesistente)
RPL_WHOREPLY (risposta_risposta al messaggio "WHO":)
RPL_ENDOFWHO (risposta_fine della risposta al messaggio "WHO":)

Esempi:

WHO *.fi ; lista tutti gli utenti coerenti con "*.fi".

WHO jto* o ; lista tutti gli utenti coerenti con "jto*" se sono operatori.

4.5.2 Richiesta Whois

Comando: WHOIS
Parametri: [<server>] <schema-nick>[,<schema-nick>[,...]]

Questo messaggio è usato per chiedere informazioni su un particolare utente. Il server risponderà a questo messaggio con diverse risposte numeriche indicando i differenti statu tra gli utenti coerenti con lo schema-nick, di tutti quelli visibili al richiedente. Se non ci sono caratteri-jolly nel parametro <schema-nick>, ogni informazione su quel nick che il richiedente sia autorizzato a vedere, viene mostrata. Può essere fornita una lista di schemi-nick separati da una virgola (','). La versione più recente invia la richiesta ad uno specifico server. Questo perchè l'attuale periodo di inattività di un utente è conosciuto solo al server al quale l'utente è direttamente connesso, mentre tutte le altre informazioni sono note universalmente. (13)

Risposte numeriche:

ERR_NOSUCHSERVER (errore_server inesistente)

ERR_NONICKNAMEGIVEN (errore_non è stato fornito alcun nick)
ERR_NOSUCHNICK (errore_nick inesistente)
RPL_WHOISUSER (risposta_WHOIS: utente)
RPL_WHOISCHANNELS (risposta_WHOIS: canali)
RPL_WHOISSERVER (risposta_WHOIS: server)
RPL_WHOISOPERATOR (risposta_WHOIS: operatore)
RPL_WHOISIDLE (risposta_WHOIS: periodo di inattività)
RPL_AWAY (risposta_utente "fuori stanza")
RPL_ENDOFWHOIS (risposta_fine del WHOIS)

Esempi:

WHOIS wiz ; ritorna le informazioni disponibili dell'utente con nick WiZ

WHOIS eff.org trillian ; chiede al server eff.org informazioni sull'user trillian

4.5.3 Domanda Whowas

Comando: WHOWAS
Parametri: <nick> [<conto> [<server>]]

Whowas chiede informazioni su un nick che non esiste più a causa di un cambio di nick oppure a causa dell'uscita da IRC dell'utente. In risposta a questa richiesta, il server ricerca nel suo archivio storico dei nick, cercando i nick che sono lessicalmente uguali (non si usano caratteri-jolly in questo caso). L'archivio storico è controllato a ritroso e fornisce per prima la ricorrenza più recente per il nick. Se viene trovata più di una presenza, vengono fornite più risposte, fino a un numero <conto> di casi (oppure la casistica completa, se non viene precisato il parametro <conto>). Se il parametro <conto> contiene un numero non positivo, allora viene effettuata una ricerca completa.

Risposte numeriche:

ERR_NONICKNAMEGIVEN (errore_non è stato fornito alcun nick)
ERR_WASNOSUCHNICK (errore_nick inesistente)
RPL_WHOWASUSER (risposta_WHOWAS: utente)
RPL_WHOWASERVER (risposta_WHOWAS: server)
RPL_ENDOFWHOWAS (risposta_fine del WHOWAS)

Esempi:

WHOWAS Wiz ; ritorna tutte le informazioni storiche sul nick "WiZ";

WHOWAS Mermaid 9 ; ritorna informazioni su un numero massimo di 9 ricorrenze più recenti per il nick Mermaid nell'archivio storico per i nicks.

WHOWAS Trillian 1 *.edu ; ritorna l'informazione storica più recente per "Trillian" dal primo server coerente con lo schema: "*.edu".

4.6 Altri Messaggi

I messaggi di questa categoria non rientrano in nessuna di quelle sopraelencate, ma sono parte integrante del protocollo.

4.6.1 Messaggio Kill

Comando: KILL
Parametri: <nick> <commento>

Il messaggio KILL viene usato per far chiudere, al server locale che la supporta materialmente, una connessione client-server. KILL viene usato dai server quando incontrano una presenza duplice nella lista di nick validi, per rimuovere ambedue gli utenti. Questo comando è disponibile anche agli operatori.

I client che possiedono algoritmi per la riconnessione automatica rendono di fatto questo comando poco utile, dal momento che la sconnessione risulta breve. Comunque esso interrompe il flusso dei dati e può essere impiegato per fermare un uso scorretto di grandi quantità di informazioni. Ogni utente può scegliere di ricevere i messaggi KILL generati da altri per tenere d'occhio i potenziali aree problematiche. In una arena dove è necessario che i nick siano in ogni momento tassativamente unici, messaggi KILL vengono generati ogniqualvolta vengono scoperti dei "duplicati" (tentativo di registrare due utenti con lo stesso nick) nella speranza che ambedue spariscano e ne riappaia solo uno. Il <commento> fornito deve riflettere la ragione effettiva del KILL. Per i KILL generati da server ,commento> è generalmente costruito con i dettagli relativi all'origine dei nick in conflitto. Per quelli generati dagli utenti è lasciato ad essi l'onere di fornire una ragione adeguata che soddisfi chi vede il comando. Per prevenire/scoraggiare la creazione di KILL truccati, per nascondere l'identità del KILLer (generatore del comando), il <commento> mostra anche un 'kill-path' (traccia del KILL) che viene aggiornato da ogni server che esso attraversa.

L'aggiornamento consiste nell'aggiunta, da parte di ogni server, del proprio nome alla traccia.

Risposte numeriche:

ERR_NOPRIVILEGES (errore_non hai i privilegi dell'operatore)
ERR_NEEDMOREPARAMS (errore_necessitano più parametri)
ERR_NOSUCHNICK (errore_nick inesistente)
ERR_CANTKILLSERVER (errore_non può "uccidere" un server)

Esempio:

KILL David (csd.bu.edu <- tolsun.oulu.fi) ; collisione di nickname tra csd.bu.edu e tolson.oulu.fi

Nota: è chiaramente auspicabile che solo gli operatori siano abilitati a chiudere una connessione di altri utenti con il messaggio KILL.

In un mondo ideale nemmeno gli operatori avrebbero bisogno di farlo, lasciando che fossero i server ad occuparsi del problema.

4.6.2 Messaggio Ping

Comando: PING
Parametri: <server1> [<server2>]

Il messaggio PING è usato per controllare la presenza di un utente attivo all'altro capo della connessione. Un messaggio PING viene mandato ad intervalli regolari, se non viene rilevata nessuna attività proveniente da una connessione. Se una connessione non risponde ad un comando PING, entro un certo periodo di tempo, quella connessione viene chiusa.

Qualsiasi client riceva un messaggio PING deve rispondere al <server1> (server che ha inviato il messaggio PING) il più in fretta possibile con un appropriato messaggio PONG, per indicare che esso è presente e vivo. I server non dovrebbero rispondere ai comandi PING ma contare sui PING dall'altro capo della connessione per indicare che la connessione è attiva. Se viene specificato il parametro <server2>, il messaggio PING viene inoltrato anche a quel server.

Risposte numeriche:

ERR_NOORIGIN (errore_nessuna origine)
ERR_NOSUCHSERVER (errore_server inesistente)

Esempi:

PING tolsun.oulu.fi ; il server che invia un messaggio PING ad un altro server per indicare che è ancora attivo.

PING WiZ ; messaggio PING inviato al nick WiZ

4.6.3 Messaggio Pong
Comando: PONG
Parametri: <demone> [<demone2>]

Il messaggio PONG è una risposte al messaggio ping. Se viene fornito il parametro <demone2> questo messaggio deve essere inoltrato anche a quel demone. Il parametro <demone> è il nome del demone che ha risposto al messaggio PING e ha generato questo messaggio.

Risposte numeriche:

ERR_NOORIGIN (errore_nessuna origine)
ERR_NOSUCHSERVER (errore_server inesistente)

Esempi:>

PONG csd.bu.edu tolsun.oulu.fi ; messaggio PONG da csd.bu.edu a tolsun.oulu.fi

4.6.4 Messaggio Error
Comando: ERROR
Parametri: <messaggio d'errore>

L'uso del comando ERROR è riservato ai server per riportare ai propri operatori un errore grave oppure fatale. Esso può anche essere inviato da un server all'altro ma non deve essere accettato se proveniente da un qualsiasi normale client non meglio conosciuto. Il messaggio ERROR è usato solamente per rendere noti gli errori che si manifestano in un link server-a-server. Un messaggio ERROR viene mandato al server che si trova all'altro capo della connessione (il quale lo invia a tutti i suoi operatori connessi) come a tutti gli operatori attualmente connessi. Il messaggio non deve essere passato ad altri server da un server che lo abbia ricevuto da un server. Quando un server invia un messaggio ERROR ricevuto ai suoi operatori, il messaggio dovrebbe essere contenuto in un messaggio NOTICE, indicando che il client non é responsabile per quell'errore.

Risposte numeriche:

Nessuna.

Esempi:

ERROR :Server *.fi already exists ; messaggio ERROR all'altro server che ha causato questo errore.

NOTICE WiZ :ERROR from csd.bu.edu -- Server *.fi already exists ; stesso messaggio ERROR come sopra, ma mandato all'utente WiZ sull'altro server.

5. MESSAGGI FACOLTATIVI
Questa sezione descrive i messaggi FACOLTATIVI. Essi non sono richiesti in una implementazione server attiva del protocollo descritto in questo documento. Se il trattamento di un messaggio opzionale non è implementato, deve essere generato un apposito messaggio di errore oppure un errore generico di comando sconosciuto. Se il messaggio è destinato ad un altro server deve essere passato oltre (è richiesta una analisi elementare). Le risposte numeriche adatte allo scopo sono elencate con i messaggi descritti qui di seguito.

5.1 Messaggio Away
Comando: AWAY
Parametri: [messaggio]

Con il messaggio AWAY i client possono predisporre una riga di risposta automatica per ogni comando PRIVMSG loro indirizzato (non ad un canale sul quale essi si trovano).

La risposta automatica viene inviata dal server al client mittente del comando PRIVMSG. L'unico server a rispondere è quello sul quale il client è localmente connesso.

Il messaggio AWAY usato sia con un parametro (per predisporre un messaggio AWAY) che senza parametri (per annullare il messaggio AWAY).

Risposte numeriche:

RPL_UNAWAY (risposta_utente non in AWAY)
RPL_NOWAWAY (risposta_utente in AWAY)

Esempi:

AWAY :Gone to lunch. Back in 5 ; predispone il messaggio away in "Gone to lunch. Back in 5' (Andato a pranzo, Di ritorno in 5 minuti).

:WiZ AWAY ; disattiva il messaggio away di WiZ.

5.2 Messaggio Rehash
Comando: REHASH
Parametri: Nessuno

Il messaggio REHASH può essere usato dall'operatore per forzare il server a rileggere e riprocessare il suo file di configurazione.

Risposte numeriche:

RPL_REHASHING (risposta_rehashing in corso)
ERR_NOPRIVILEGES (errore_non hai i privilegi dell'operatore)

Esempi:

REHASH ; messaggio da un client, con status operatore, ad un server per chiedergli di rileggere il suo file di configurazione.

5.3 Messaggio Restart
Comando: RESTART
Parametri: Nessuno

Il comando RESTART può essere usato esclusivamente da un operatore per forzare un server ad attivare la procedura di riavvio. Questo messaggio è facoltativo in quanto potrebbe essere visto come un rischio permettere a persone non meglio qualificate di connettersi al server come operatori ed eseguire questo comando, causando (come minimo) un; interruzione del servizio. Il comando RESTART deve sempre essere processato completamente dal server al quale il client mittente è connesso, e non deve essere inoltrato ad altri server.

Risposte numeriche:

ERR_NOPRIVILEGES (errore_non hai i privilegi dell'operatore)

Esempi:

RESTART ; non sono richiesti parametri.

5.4 Messaggio Summon
Comando: SUMMON
Parametri: <utente> [<server>]

Il comando SUMMON può essere usato per inviare, ad utenti che si trovano su una macchina su cui è attivo un server IRC, un messaggio per chiedere loro di unirsi a IRC. Questo messaggio mandato solamente se il server destinazione del messaggio:

a) ha il comando SUMMON abilitato;
b) l'utente è in linea;
c) il server che processa il comando può inviare dell'output all'utente.

Se non viene dato alcun parametro <server>, il server cerca di SUMMON (convocare) l'<utente> sul server sul quale il client è connesso che viene considerato come server destinazione.

Se SUMMON non è attivo su un server, esso deve mandare la risposta numerica ERR_SUMMONDISABLED e passare oltre il messaggio. (14) Risposte numeriche:

ERR_NORECIPIENT (errore_nessun destinatario)
ERR_FILEERROR (errore_errore sul file)
ERR_NOLOGIN (errore_non in linea)
ERR_NOSUCHSERVER (errore-server inesistente)
RPL_SUMMONING (risposta_convocazione in corso)

Esempi:

SUMMON jto ; convoca l'utente jto sulla macchina del server

SUMMON jto tolsun.oulu.fi ; convoca l'utente jto sulla macchina sulla quale è attivo il server "tolsun.oulu.fi".

5.5 Comando Users, Utenti
Comando: USERS
Parametri: [<server>]

Il comando USERS invia in risposta una lista di utenti collegati ad un server in un formato simile a quello di who(1), rusers(1) e finger(1). Alcuni disabilitano questo comando dal loro server per ragioni di sicurezza. Se disabilitato, la relativa risposta numerica, indicante la disabilitazione, deve essere fornita.

Risposte numeriche:

ERR_NOSUCHSERVER (errore_server inesistente)
ERR_FILEERROR (errore_errore sul file)
ERR_USERSDISABLED (errore_comando USERS disabilitato)
RPL_USERSSTART (risposta_USERS: inizio)
RPL_USERS (risposta_USERS)
RPL_NOUSERS (risposta_nessun utente)
RPL_ENDOFUSERS (risposta_fine degli utenti)

Risposta se disabilitato:

ERR_USERSDISABLED (errore_comando USERS disabilitato)

Esempi:

USERS eff.org ; richiede una lista di utenti connessi su eff.org

:John UTENTI tolsun.oulu.fi ; richiesta di John per una lista di utenti connessi sul server tolsun.oulu.fi

5.6 Comando Operwall
Comando: WALLOPS
Parametri: Testo da mandare a tutti gli operatori in linea

Manda un messaggio a tutti gli operatori in linea. Dopo aver implementato il WALLOPS come un comando a disposizione del generico utente, ci si è accorti che di esso veniva spesso e volentieri fatto un uso improprio per mandare messaggi ad un sacco di gente (in maniera molto simile al WALL).

Per questo motivo sarebbe bene che la corrente implementazione del comando WALLOPS sia considerata come un esempio e che siano in futuro riconosciuti solo i server come mittenti dei WALLOPS.

Risposte numeriche:

ERR_NEEDMOREPARAMS (Errore_necessitano più parametri)

Esempi:

:csd.bu.edu WALLOPS :Connect '*.uiuc.edu 6667' from Joshua ; messaggio WALLOPS da csd.bu.edu che annuncia un messaggio CONNECT ricevuto da Joshua e messo in atto.

5.7 Messaggio Userhost
Comando: USERHOST
Parametri: <nick>{<carattere-spazio><nickname>}

Il comando USERHOST prende una lista contenente fino a 5 nick separati da un carattere spazio, e invia una lista di informazioni su ogni nick che ha trovato. La lista ha le risposte separate da uno spazio.

Risposte numeriche:

ERR_NEEDMOREPARAMS (Errore_necessitano più parametri)
RPL_USERHOST (risposta_USERHOST)

Esempi:

USERHOST Wiz Michael Marty p ; Richiesta di informazioni USERHOST sui nick "Wiz", "Michael", "Marty" e "p"

5.8 Messaggio ISON
Comando: ISON
Parametri: <nick>{<carattere-spazio><nick>}

Il comando ISON è stato implementato per fornire un mezzo veloce ed efficiente per sapere se un dato nick è correntemente presente su IRC.

ISON prevede solo un (1) parametro: una lista di nick separati da spazi. Ogni nick della lista che sia attualmente presente, viene aggiunto dal server alla stringa contenente le risposte. Così la stringa di risposta può essere vuota (nessuno dei nick presente), oppure viene inviata una copia esatta della stringa dei parametro (tutti i nick presenti) o qualsiasi altra sottoinsieme dei nick dati nel parametro. L'unico limite sul numero di nick che può essere controllato, è dato dalla lunghezza della riga che non deve eccedere i 512 caratteri, altrimenti verrà troncata dal server. ISON è processato solamente dal server locale del client che manda il comando e non viene passato ad altri servers.

Risposte numeriche:

ERR_NEEDMOREPARAMS (Errore_necessitano più parametri)
RPL_ISON (risposta_utente in linea)

Esempi:

ISON phone trillian WiZ jarlek Avalon Angel Monstah ; esempio di richiesta ISON per 7 nicks.

6. RISPOSTE

Quella che segue è una lista di risposte numeriche che sono generate in risposta ai comandi descritti sopra. Ogni risposta numerica è data con il suo numero, il nome e la relativa stringa esplicativa.

6.1 Risposte d'Errore

401 ERR_NOSUCHNICK "<nickname> :No such nick/channel"
Risposta inviata per indicare che il parametro nickname dato ad un comando è attualmente non in uso.

402 ERR_NOSUCHSERVER "<server name> :No such server"
Risposta inviata per indicare che il nome del server dato attualmente non esiste.

403 ERR_NOSUCHCHANNEL "<channel name> :No such channel"
Risposta inviata per indicare che il nome del canale fornito non è valido.

404 ERR_CANNOTSENDTOCHAN "<channel name> :Cannot send to channel"
Risposta inviata ad un utente il quale: a) non si trova su un canale con mode +n, oppure b) non è un operatore di canale (o non è in mode utente +v) su un canale in mode +m e sta cercando di inviare un PRIVMSG a quel canale.

405 ERR_TOOMANYCHANNELS "<channel name> :You have joined too many channels"
Risposta inviata agli utenti quando si sono aggregati al numero massimo permesso di canali, e stanno cercando di aggregarsi ad un altro canale.

406 ERR_WASNOSUCHNICK "<nickname> :There was no such nickname"
Mandata in risposta ad un WHOWAS per indicare che non ci sono informazioni nell'archivio storico per quel nick.

407 ERR_TOOMANYTARGETS "<target> :Duplicate recipients. No message\delivered"
Replica inviata in risposta ad un client che sta cercando di mandare un PRIVMSG/NOTICE usando il formato di destinazione user@host e per un user@host che ha diverse presenze al momento attuale.

409 ERR_NOORIGIN ":No origin specified"
Messaggio PING o PONG al quale manca il parametro che indica il mittente, necessario dal momento che questi comandi devono funzionare senza prefissi validi.

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 sono date da PRIVMSG per indicare che il messaggio non è stato inoltrato per qualche ragione. ERR_NOTOPLEVEL e ERR_WILDTOPLEVEL sono errori che vengono inviati come risposta quando si tenta un uso non valido di "PRIVMSG $<server>" o "PRIVMSG #<host>".

421 ERR_UNKNOWNCOMMAND "<Command> :Unknown Command"
Risposta inviata ad un client registrato per indicare che il comando proposto è sconosciuto al server.

422 ERR_NOMOTD ":MOTD File is missing"
Il server non ha potuto aprire il proprio file MOTD. [Ndr - MOTD = Message Of The Day (messaggio del giorno)]

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

Risposta inviata da un server in risposta ad un messaggio ADMIN quando c'è un errore nel reperimento delle informazioni.

424 ERR_FILEERROR ":File error doing <file op> on <file>"
Messaggio di errore generico, usato per informare il fallimento di un operazione di trattamento di file(s) durante il trattamento del messaggio.

431 ERR_NONICKNAMEGIVEN ":No nickname given"
Risposta inviata quando un parametro nickname richiesto per l'esecuzione di un comando non viene trovato

432 ERR_ERRONEUSNICKNAME "<nick> :Erroneus nickname"
Risposta inviata dopo aver ricevuto un messaggio NICK il quale contiene caratteri non compresi nel set definito per i nicks. Si veda la sezione 4.1.2 per dettagli sui nickname validi.

433 ERR_NICKNAMEINUSE "<nick> :Nickname is already in use"
Risposta inviata quando un messaggio NICK ha come risultato un tentativo di cambiare il nickname con un nickname già correntemente in uso.

436 ERR_NICKCOLLISION "<nick> :Nickname collision KILL"
Risposta inviata da un server ad un client quando scopre una collisione di nickname (registrazione di un nick che già esiste da parte di un altro server).

441 ERR_USERNOTINCHANNEL "<nick> <channel> :They aren't on that channel"
Risposta inviata dal server per indicare che l'utente destinatario del comando non è sul canale dato.

442 ERR_NOTONCHANNEL "<channel> :Yoùre not on that channel"
Risposta inviata dal server ogni volta che un client tenta un comando riferito ad un canale del quale non è membro.

443 ERR_USERONCHANNEL "<user> <channel> :is already on channel"
Risposta inviata quando un client cerca di invitare un utente nel canale nel quale già si trovano ambedue.

444 ERR_NOLOGIN "<user> :User not logged in"
Risposta inviata dopo che il comando SUMMON per un utente non stato portato a termine in quanto l'utente non era in linea.

445 ERR_SUMMONDISABLED ":SUMMON has been disabled"
Risposta inviata come risposta al comando SUMMON. Deve essere inviata da ogni server sul quale questo comando non è implementato.

446 ERR_USERSDISABLED ":USERS has been disabled"
Risposta inviata in risposta al comando USERS. Deve essere inviata da ogni server sul quale questo comando non è implementato.

451 ERR_NOTREGISTERED ":You have not registered"
Risposta inviata dal server per indicare che il client deve essere registrato prima che il server permetta ad esso di essere analizzato nei dettagli.

461 ERR_NEEDMOREPARAMS "<Command> :Not enough parameters"
Risposta inviata dal server in risposta a numerosi comandi per indicare che il client non ha fornito un numero di parametri sufficiente.

462 ERR_ALREADYREGISTRED ":You may not reregister"
Risposta inviata dal server per ogni connessione che cerchi di cambiare parte delle notizie registrate (quali password o informazioni concernenti l'utente) mediante un secondo comando USER.

463 ERR_NOPERMFORHOST ":Your host isn't among the privileged"
Risposta inviata ad un client che cerca di registrarsi su un server che non permette connessioni dalla macchina che tenta la connessione.

464 ERR_PASSWDMISMATCH ":Password incorrect"
Risposta inviata per indicare il fallimento di un tentativo di registrare una connessione per la quale è richiesta una parola d'ordine e quest'ultima non è stata fornita oppure è stata indicata in forma non corretta.

465 ERR_YOUREBANNEDCREEP ":You are banned from this server"
Risposta inviata dopo un tentativo di connessione e registrazione su un server predisposto per negarci esplicitamente la connessione.

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- Yoùre not an IRC operator"
Ogni comando che richiede i privilegi di operatore per essere eseguito, deve inviare questa risposta d'errore per indicare che il tentativo di eseguire il comando non ha avuto successo.

482 ERR_CHANOPRIVSNEEDED "<channel> :Yoùre not channel operator"
Ogni comando che richiede i privilegi di operatore di canale per essere eseguito (quale, ad esempio, il messaggio MODE) deve mandare questo errore se il client sta facendo un tentativo di eseguire il comando e non è un operatore sul canale specificato.

483 ERR_CANTKILLSERVER ":You cant kill a server!"
Ogni tentativo di usare il comando KILL su un server deve essere rifiutato e questo messaggio d'errore deve essere inviato direttamente al client.

491 ERR_NOOPERHOST ":No O-lines for your host"
Se un client invia un messaggio OPER ed il server non è stato configurato per permettere connessioni dalla macchina del client come operatore, deve essere inviata questa risposta d'errore.

501 ERR_UMODEUNKNOWNFLAG ":Unknown MODE flag"
Risposta inviata dal server per indicare che un messaggio MODE è stato inviato con un parametro nickname e che la modalità mode inviata non è stata riconosciuta.

502 ERR_USERSDONTMATCH ":Can't change mode for other users"
Errore Inviato a qualsiasi utente che cerchi di prender visione o cambiare gli user mode per un utente che non sia se stesso.

6.2 Risposte ai comandi.
300 RPL_NONE Replica non usata.

302 RPL_USERHOST ":[<reply>{<space><reply>}]"
Formato di replica usato da USERHOST per elencare risposte alla lista che compare nella richiesta. La stringa di replica è composta come segue:

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

L'asterisco '*' indica se un client è registrato come operatore. I caratteri '-' o '+' indicano rispettivamente se il client ha predisposto oppure no un messaggio di AWAY.

303 RPL_ISON ":[<nick> {<space><nick>}]"
Formato di replica usato da ISON per elencare risposte alla lista che compare nella richiesta.

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"
Queste repliche sono usate con il comando AWAY (se permesso). RPL_AWAY è inviata ad ogni client che invii un messaggio ad un client che abbia settato il comando AWAY. RPL_AWAY è inviata solo dal server al quale il client è connesso. Le repliche RPL_UNAWAY e RPL_NOWAWAY sono inviate quando il client annulla o predispone il messaggio AWAY.

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>}"
Le repliche dalla 311 alla 313 e dalla 317 alla 319 sono tutte generate in risposta ad un messaggio WHOIS. Ammesso che sia presente un numero di parametri sufficiente, il server che risponde deve formulare o una risposta numerica tra quelle sopra elencate (nel caso che il nick richiesto venga trovato), oppure deve inviare una risposta di errore. L'asterisco '*' in RPL_WHOISUSER è in questo caso inserito come carattere e non come wild card. Per ogni insieme di risposte, solamente RPL_WHOISCHANNELS può apparire più di una volta (per liste lunghe di nomi di canali). Il caratteri '@' e '+' affiancati al nome del canale indicano se un client è operatore di quel canale o se gli è stato accordato il permesso di parlare su un canale moderato. La replica RPL_ENDOFWHOIS è usata per segnalare la fine del trattamento del messaggio WHOIS.

314 RPL_WHOWASUSER "<nick> <user> <host> * :<real name>"

369 RPL_ENDOFWHOWAS "<nick> :End of WHOWAS"
Quando risponde ad un messaggio WHOWAS, il server deve usare le repliche RPL_WHOWASUSER, RPL_WHOISSERVER o ERR_WASNOSUCHNICK per ogni nickname della lista presentata. Alla fine di ogni gruppo di risposte, deve comparire un RPL_ENDOFWHOWAS (anche se c'è stata una sola risposta e questa era un errore).

321 RPL_LISTSTART "Channel :Users Name"

322 RPL_LIST "<channel> <# visible> :<topic>"

323 RPL_LISTEND ":End of /LIST"
Le repliche RPL_LISTSTART, RPL_LIST, RPL_LISTEND segnano rispettivamente l'inizio, le vere e proprie risposte con dati, e la fine delle repliche del server al comando LIST. Se non ci sono canali disponibili sui quali ritornare notizie, devono essere inviate solo le repliche di inizio e fine.

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

331 RPL_NOTOPIC "<channel> :No topic is set"

332 RPL_TOPIC "<channel> :<topic>"
Quando si manda un messaggio TOPIC per definire il topic del canale, viene inviata una delle due repliche. SE il topic è definito viene Inviata la replica RPL_TOPIC, diversamente viene Inviata la replica RPL_NOTOPIC.

341 RPL_INVITING "<channel> <nick>"
Risposta inviata dal server per indicare che il tentativo INVITE è stato accettato e quindi è stato segnalato al client.

342 RPL_SUMMONING "<user> :Summoning user to IRC"
Risposta inviata da un server in risposta ad un messaggio SUMMON per indicare che sta convocando quell'utente.

351 RPL_VERSION "<version>.<debuglevel> <server> :<comments>"
Risposta inviata dal server per fornire informazioni di tipo VERSION.
Il parametro <version> è la versione del software in uso (incluse tutte le revisioni ed i livelli di aggiornamento) e il parametro <debuglevel> è usato per indicare se il server lavora in "debug mode" [Ndr - Modo diagnostico]. Il campo "comments" può contenere commenti sulla versione oppure ulteriori notizie sulla versione.

352 RPL_WHOREPLY "<channel> <user> <host> <server> <nick>\<H|G>[*][@|+] :<hopcount> <real name>"

315 RPL_ENDOFWHO "<name> :End of /WHO list"
La coppia di repliche RPL_WHOREPLY e RPL_ENDOFWHO sono usate per rispondere ad un messaggio WHO. La replica RPL_WHOREPLY è inviata solamente se è stato rintracciato un riscontro appropriato alla domanda WHO. Se viene data una lista di parametri a supporto del messaggio WHO message, deve essere inviata una replica RPL_ENDOFWHO dopo aver processato ogni voce della lista, intendendo per voce il parametro <name>.

353 RPL_NAMREPLY "<channel> :[[@|+]<nick> [[@|+]<nick>[...]]]"

366 RPL_ENDOFNAMES "<channel> :End of /NAMES list"
Per rispondere ad un messaggio NAMES, viene inviata una coppia di repliche composta da RPL_NAMREPLY e RPL_ENDOFNAMES dal server al client. Se non viene trovato alcun canale, tra quelli dati con la domanda, allora viene inviata solamente la replica RPL_ENDOFNAMES. L'eccezione si dà quando il messaggio NAMES viene inviato senza parametri e tutti i canali visibile ed i loro contenuti, sono inviati in una serie di repliche RPL_NAMEREPLY con RPL_ENDOFNAMES a segnarne la fine.

364 RPL_LINKS "<mask> <server> :<hopcount> <serverinfo>"

365 RPL_ENDOFLINKS "<mask> :End of /LINKS list"
In risposta al messaggio LINKS, un server deve inviare delle repliche usando la risposta numerica RPL_LINKS, segnando la fine della lista usando la replica RPL_ENDOFLINKS.

367 RPL_BANLIST "<channel> <banid>"

368 RPL_ENDOFBANLIST "<channel> :End of channel ban list"
Quando elenca i 'ban' attivi per un dato canale, al server si richiede di inviare una lista usando le repliche RPL_BANLIST e RPL_ENDOFBANLIST. Un messaggio separato RPL_BANLIST è inviato per ogni ban attivo. Dopo che i ban sono stati elencati (oppure se non ne era prendente nessuno) deve essere inviata la replica RPL_ENDOFBANLIST.

371 RPL_INFO ":<string>"

374 RPL_ENDOFINFO ":End of /INFO list"
Un server che risponde ad un messaggio INFO deve essere in grado di mandare tutte le sue 'informazionì in una serie di repliche RPL_INFO con la replica RPL_ENDOFINFO ad indicare la fine della serie di risposte.

375 RPL_MOTDSTART ":- <server> Message of the day - "

372 RPL_MOTD ":- <text>"

376 RPL_ENDOFMOTD ":End of /MOTD Comando"
Quando viene trovato il file MOTD in risposta al messaggio MOTD, il file viene mostrato riga per riga, ognuna delle quali non è più lunga di 80 caratteri, usando il formato di replica RPL_MOTD. Questa replica (RPL_MOTD) dovrebbe essere preceduta da RPL_MOTDSTART e seguita da RPL_ENDOFMOTD.

381 RPL_YOUREOPER ":You are now an IRC operator"

La replica RPL_YOUREOFOR è inviata ad un client che ha espletato con successo un messaggio OPER e guadagnato lo status operatore.

382 RPL_REHASHING "<config file> :Rehashing"
Se viene usata l'opzione REHASH ed un operatore invia un messaggio REHASH, a quell'operatore viene inviata la replica RPL_REHASHING.

391 RPL_TIME "<server> :<string showing server's local time>"
Quando un server risponde ad un messaggio TIME, deve inviare la risposta usando il format RPL_TIME descritto sopra. La stringa che mostra l'orario deve contenere solamente il giorno e l'orario corretti. Non sono necessari altri requisiti.

392 RPL_USERSSTART ":UserID Terminal Host"

393 RPL_USERS ":%-8s %-9s %-8s"

394 RPL_ENDOFUSERS ":End of users"

395 RPL_NOUSERS ":Nobody logged in"
Quando il messaggio USER è trattato da un server, sono usate le repliche RPL_USERSTART, RPL_USERS, RPL_ENDOFUSERS e RPL_NOUSERS. La replica RPL_USERSSTART deve essere inviata per prima, seguita o da una sequenza di RPL_USERS, oppure da una singola RPL_NOUSER. Per ultima deve essere inviata una RPL_ENDOFUSERS.

200 RPL_TRACELINK "Link <version & debug level> <destination> \ <next server>"

201 RPL_TRACECONNECTING "Try. <class> <server>"

202 RPL_TRACEHESHAKE "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>"
Le repliche RPL_TRACE* sono tutte inviate dal server in risposta ad un messaggio TRACE. Quante ne vengano inviate dipende dal messaggio TRACE e se questo sia stato inviato da un operatore oppure da un normale utente. Non c'è un ordine predefinito per quale replica debba essere inviata per prima. Le repliche RPL_TRACEUNKNOWN, RPL_TRACECONNECTING e RPL_TRACEHESHAKE sono tutte usate per connessioni che non sono state stabilite in maniera completa e risultano sconosciute, oppure che sono ancora in corso di connessione, oppure stanno completando il processo di 'server handshake'. La replica RPL_TRACELINK è inviata da qualsiasi server che tratta un messaggio TRACE e deve passarlo ad un altro server. La lista di RPL_TRACELINKs inviata in risposta ad un comando TRACE che attraversa la rete IRC dovrebbe riflettere l'assetto effettivo delle connessioni tra i server lungo quel percorso. La replica RPL_TRACENEWTYPE deve essere usata per ogni connessione che non rientra nelle altre categorie ma viene comunque mostrata.

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>"
Per rispondere ad una domanda sul mode di un client viene inviata la replica RPL_UMODEIS.

251 RPL_LUSERCLIENT ":There are <integer> user 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 e <integer> \ servers"
Mentre processa un messaggio LUSERS, il server invia degli insiemi di risposte dalla 251 alla 255. Mentre risponde, il server deve inviare RPL_LUSERCLIENT e RPL_LUSERME. Le altre risposte vengono inviate solamente se viene trovato per loro un conteggio diverso da 0.

256 RPL_ADMINME "<server> :Administrative info"

257 RPL_ADMINLOC1 ":<admin info>"

258 RPL_ADMINLOC2 ":<admin info>"

259 RPL_ADMINEMAIL ":<admin info>"
Quando risponde ad un messaggio ADMIN, il server è tenuto ad usare le repliche (dalla 256 alla 259) elencate qui sopra e fornire un messaggio di testo per ogni replica. Per la replica RPL_ADMINLOC1 il server deve dare una descrizione di: città, stato e paese in cui esso si trova, seguita da informazioni sull'università e il dipartimento (RPL_ADMINLOC2) e, per ultimo, il contatto amministrativo per il server (è richiesto un indirizzo e-mail) nella risposta RPL_ADMINEMAIL. (15)

6.3 Risposte numeriche riservate.
Queste risposte numeriche non state descritte prima in quanto esse rientrano in una delle seguenti categorie:

1. non più in uso
2. riservate per un prevedibile uso futuro
3. correntemente in uso ma parte di un assetto prestazionale non generale dell'attuale server IRC.

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. AUTENTICAZIONE DI CLIENT E SERVER

I client e i server sono soggetti allo stesso livello di autenticazione.

Per entrambi, per ogni connessione fatta al server, viene effettuato un passaggio dall'IP al nome della macchina (ed un controllo all'inverso su questo). Entrambe le connessioni sono poi soggette ad un controllo sulla parola d'ordine (se ne è stata definita una per quella connessione).

Questi controlli sono possibili su tutte le connessioni sebbene il controllo sulla parola d'ordine sia usato, in generale, solamente con i server.

Un controllo addizionale, che sta diventando sempre più usuale, è il controllo sull'username (nome dell'utente) responsabile della connessione. Trovare lo username dall'altra parte connessa comporta la connessione ad un server di autenticazione quale IDENT, come viene descritto nel documento RFC 1413.

Dato che senza parole d'ordine non è cosa facile determinare con sicurezza chi si trovi all'altro capo di una connessione, l'uso delle parole d'ordine è fortemente consigliabile nelle connessioni tra server in aggiunta a misure di altra natura quale l'uso di un ident server.

8. Implementazioni attuali

L'unica implementazione corrente di questo protocollo è IRC server versione 2.8. [Ndr - Naturalmente questa affermazione era valida al momento della pubblicazione di questo documento nel maggio 1993: già al momento di questa traduzione (giugno 1997) non lo è più]. (16) Versioni precedenti potevano implementare in parte o tutti i comandi descritti in questo documento, sostituendo molte delle risposte numeriche con messaggi NOTICE. Purtroppo, per esigenze di compatibilità verso versioni precedenti, l'implementazione di alcune parti di questo documento non è conforme al modo in cui era stata progettata. Una differenza notevole:

* il riconoscimento che ogni LF o CR che si trovino in un qualunque punto di un messaggio marchino la fine di quel messaggio (invece di richiedere il CR-LF)

Il resto di questa sezione tratta argomenti che hanno valore soprattutto per coloro i quali desiderino implementare un server, ma alcune parti del protocollo trovano un'utile e diretta applicazione anche ai clients.

8.1 Protocollo Network: TCP perchè esso trova qui un applicazione ottimale.

L'IRC è stato implementato sulla base del TCP, dal momento che il TCP fornisce un protocollo di rete affidabile e che ben si adatta al tipo e dimensione del sistema di scambio di messaggi cui IRC appartiene. L'uso di IP multidirezionale costituisce un alternativa, ma non è disponibile su vasta scala al momento.

8.1.1 Sostegno per Unix sockets

Dato che il campo d'azione degli Unix sockets permette operazioni ascolto/connessione, l'implementazione corrente può essere configurata in modo che essa ascolti ed accetti le connessioni sia da client che da server su un socket di dominio Unix. Queste connessioni vengono riconosciute come socket quando il nome dell'host inizia con il carattere '/'.

Quando il server fornisce delle informazioni sulle connessioni socket di dominio Unix, esso deve sostituire l'host name effettivo con il pathname, a meno che non sia richiesto proprio il nome effettivo del socket.

8.2 Analisi dei comandi

Per fornire un valido I/O (Input/Output) 'non-buffered' in rete, sia per i client che per i servers, ad ogni connessione viene assegnato un 'input buffer', dedicato, nel quale sono memorizzati i risultati delle operazioni di lettura e delle analisi di comandi più recenti. è stata stabilita per il buffer un ampiezza di 512 byte in modo che possa contenere un messaggio completo, anche se, normalmente, contiene più di un comando per volta. Il buffer dedicato viene analizzato dopo ogni operazione di lettura per verificare la presenza di comandi validi. Quando accade di dover trattare nel buffer messaggi multipli da un client, bisogna usare attenzione qualora un messaggio causi la rimozione del client.

8.3 Invio del messaggio

É cosa molto comune trovare link della rete saturi oppure macchine, alle quali si inviano dati, incapaci di inviare dati. Sebbene Unix tratti questo problema con la finestra TCP e buffer interni, il server ha spesso enormi quantità di dati da inviare (specialmente quando si forma un nuovo link server-server) ed i piccoli buffer forniti nel kernel del sistema operativo, non sono sufficienti per la lunga coda di dati in uscita. Per alleviare questo problema viene usato una "send queue" (sequenza di invio) con i dati organizzati in una struttura FIFO. [Ndr - Acronimo di First In, First Out, un modello di organizzazione di dati] Una tipica "send queue" può arrivare fino a 200 Kbyte di ampiezza in una rete IRC di grandi dimensioni, con una connessione alla rete lenta, quando un nuovo server si connette.

Nel raccogliere dati dalle sue connessioni, un server in primo luogo leggera ed analizzerà tutte le informazioni in entrata, mettendo in coda ogni dato da inviare all'esterno. Quando tutti i dati in input disponibili sono stati trattati, la sequenza di dati preparata viene inviata. Questo riduce il numero delle chiamate di sistema al comando write() (scrittura) e permette al TCP il trattamento di pacchetti più grandi.

8.4 Connessione 'Liveness'

Per controllare quando una connessione si sia persa oppure non risponda più, il server deve fare un ping per ognuna delle sue connessioni dalle quali non riceve risposte da un dato lasso di tempo.

Se una connessione non risponde in tempo, verrà chiusa seguendo le procedure appropriate. Una connessione viene fatta cadere anche nel caso in cui la sua send queue (coda di dati da inviare) cresce al di la del massimo consentito, in quanto è senz'altro meglio chiudere una connessione lenta piuttosto che il server si blocchi.

8.5 Stabilire una connessione server-client

Quando un client si connette ad un server IRC, gli viene inviato il comando MOTD (se presente) come del resto il numero di utenti e di server attualmente connessi (come avviene per il comando LUSER). Viene inoltre richiesto al server di fornire un messaggio non ambiguo che notifichi al client il suo nome e versione ed ogni altra informazione che possa sembrare appropriata.

Dopo aver espletato queste funzioni, il server deve inviare il nickname del nuovo utente e tutte le altre informazioni fornibili da lui stesso (Comando USER), come del resto le notizie reperibili in altra sede (dai server con funzione di DNS/authentication). Il server deve inviare queste informazioni con NICK seguito da USER.

8.6 Stabilire una connessione server-server

Il procedimento usato per stabilire una connessione server-server presenta aspetti particolarmente rischiosi, in quanto ci sono innumerevoli circostanze nelle quali è molto facile che vengano commessi errori - il meno rilevante dei quali sono le cosiddette "race conditions". [Ndr - Condizioni per la corretta risoluzione delle quali la variabile tempo è di importanza essenziale]

Dopo che un server ha ricevuto una connessione seguita dalla coppia di comandi PASS/SERVER, riconosciuta

valida, il server dovrebbe rispondere con le sue informazioni PASS/SERVER per quella connessione, e con tutte le altre informazioni di stato che conosce, come viene descritto qui di seguito.

Quando il server che inizia la connessione riceve la coppia di comandi PASS/SERVER, anch'esso controlla che il server che risponde sia autenticato in maniera appropriata, prima di accettare la connessione con quel server.

8.6.1 Scambio di informazioni di stato fra server al momento della connessione

L'ordine delle informazioni di stato che vengono scambiate tra server è essenziale. L'ordine richiesto è il seguente:

* tutti gli altri server conosciuti;
* tutte le informazioni conosciute sugli utenti;
* tutti le informazioni conosciute sui canali.

Le informazioni riguardanti i server vengono inviate mediante messaggi SERVER, le informazioni concernenti gli utenti con messaggi NICK/UTENTE/MODE/JOIN e le informazioni sui canali via messaggi MODE.

Nota: i topic dei canali *NON* vengono scambiati in questa circostanza: il comando TOPIC cancella ed aggiorna ogni informazione topic precedente, cosicchè, nella migliore delle ipotesi, i due versanti della connessione cambierebbero i topics. Passando per prime le informazioni di stato dei servers, tutte le collisioni tra server che gia esistano hanno luogo prima che possano aver luogo le collisioni sui nick dovute al fatto che un secondo server introduce un nick che gia esista su un altro). Dal momento che la rete IRC è capace di esistere solo come un grafo aciclico, potrebbe essere possibile che la rete si sia gia riconnesso in un altra locazione: il luogo dove avviene la collisione tra server indica allora il luogo dove la rete necessita di uno split.

8.7 Terminazione di connessioni server-client

Quando una connessione client chiude, viene generato un messaggio QUIT, ad interesse del client, dal server al quale il client era connesso. Non è necessario generare od usare altri messaggi.

8.8 Terminazione di connessioni server-server

Se una connessione server-server viene chiusa, che sia per un messaggio remoto SQUIT sia per cause 'naturalì, il resto della rete IRC connessa deve essere informata della sconnessione dal server che ha scoperto la chiusura che invierà una lista di SQUIT (una per ogni server che si trova oltre la sconnessione) ed una lista di QUIT (una per ogni client nella stessa posizione).

8.9 Seguire la traccia dei cambi di nickname

Tutti i server IRC devono tenere una traccia storica dei cambi recenti di nick. Questo è richiesto per permettere al server di avere una possibilità di restare in contatto con la situazione quando cambi di nick si succedano a gran velocità (race conditions) insieme con comandi che trattano i nick stessi. I comandi che devono tener traccia dei cambi di nick sono:

* KILL (il nickname ha subito un KILL)
* MODE (+/- o,v)
* KICK (il nickname subisce dei KICK)

Per nessun altro comando devono essere controllati i cambi di nick.

Nei casi sopra descritti, al server è richiesto innanzi tutto di controllare l'esistenza del nick, e poi di controllarne la storia per vedere a chi appartiene al momento (sempre se qualcuno stia usando quel nickname !). Questo riduce le possibilità di trovarsi in situazioni difficoltose a causa della velocità con cui si succedono le operazioni (race conditions), ma difficoltà possono comunque verificarsi qualora il server finisca per eseguire il comando su un client sbagliato. Quando si attua il tracciamento per i cambi di nickname, come descritto qui sopra, si raccomanda che venga concesso un intervallo di tempo e che vengano ignorati casi troppo vecchi. Per avere un archivio storico ragionevole, un server dovrebbe essere in grado di tenere i

nick precedenti di ogni client che conosce, qualora tutti decidessero di cambiare. l'ampiezza dell'archivio storico è comunque limitata da fattori diversi (per esempio dalla disponibilità di memoria, ecc.).

8.10 Controllo del flood dei clients

Con una grande rete IRC di server interconnessi, è molto facile per qualsiasi client connesso alla rete, di produrre un flusso continuo di messaggi che finisce non solo per floodare [Ndr - flood vale allagamento, alluvione, inondazione] la rete, ma anche per degradare il livello delle prestazioni erogate verso gli altri clients.

Piuttosto che richiedere ad ogni possibile vittima di provvedere alla propria protezione, la protezione flood è stata scritta nel server ed applicata a tutti i client tranne che ai servizi.

L'algoritmo adottato segue il seguente schema:

* controlla se il 'message timer' segna un tempo precedente rispetto all'orario corrente (aggiornarlo, se necessario, per renderlo uguale al tempo corrente);
* leggere ogni dato presente proveniente dal client;
* mentre il timer è avanti di meno di dieci secondi dall'orario corrente, analizza ogni messaggio presente, e penalizza il client di 2 secondi per ogni messaggio;

il che significa, di fatto, che il client può mandare 1 messaggio ogni 2 secondi senza subire penalità.

8.11 Letture veloci per evitare blocchi

In un ambiente real-time, è essenziale che tutte le azioni dei server attendano il minor tempo possibile per essere espletate, cosicchè tutti i client siano serviti in modo equo. Ovviamente questo richiede un non-blocking I/O su tutte le operazioni read/write del network. Per le connessioni server normali, questo non sarebbe difficile, ma ci sono altre operazioni di supporto che possono causare il blocco temporaneo del server (quali la lettura dei dischi). Dove possibile, una tale attività dovrebbe essere espletata con pause di funzionamento brevi.

8.11.1 Lettura veloce dell'Hostname (DNS)

l'uso delle librerie di Berkley e di altre, per la risoluzione degli indirizzi, comportava lunghi tempi di elaborazione ed in alcuni casi le risposte arrivavano fuori tempo massimo. Per evitarlo sono state scritte delle routine alternative per la risoluzione dei DNS, predisposte per operazioni di I/O che non comportano pause di funzionamento, e da cui si raccolgono dati nell'ambito del loop di I/O principale del server.

8.11.2 Lettura veloce dell'Username (Ident)

Tutte le numerose librerie di identificazione che ci sono a disposizione, adatte ad essere incluse ed adoperate all'interno di altri programmi, hanno causato inconvenienti legati al fatto che operano in maniere sincrona, causando ritardi notevoli e frequenti. Ancora una volta la soluzione è stata quella di scrivere degli insiemi di routine capaci di cooperare in maniera efficiente con il resto del software dei server usando funzioni di I/O che non comportano pause di funzionamento.

8.12 File di configurazione

Per fornire un modo flessibile per configurare e far funzionare il programma server, è auspicabile l'impiego di un file di configurazione che contenga istruzioni per il server sugli aspetti seguenti:

* da quali host accettare connessioni da client;
* quali host autorizzare a connettersi come servers;
* a quali host possa connettersi (sia in modo attivo che passivo);
* informazioni sulla collocazione del server (per esempio: università, città/stato, azienda);
* chi sono i responsabili per quel server ed un indirizzo e-mail dove possano essere contattati;
* hostname e parola d ordine per quei client ai quali si desidera che sia dato accesso ai comandi riservati agli operatori.

Nello specificare gli hostname, sia il domain name (composto da caratteri alfabetici) sia l'uso della 'notazione numerica' (127.0.0.1) dovrebbero essere accettati. Deve essere possibile specificare la parola d ordine che deve essere usata / accettata per tutte le connessioni in entrata ed in uscita (malgrado le uniche connessioni in uscita siano quelle verso altri server).

La lista precedente è il minimo richiesto per un server che desideri fare una connessione con un altro server. Altre voci che potrebbero essere utili sono:

* quali server altri server possono introdurre;
* profondità permessa ad una ramificazione del server;
* l'orario durante il quale i client si possono connettere.

8.12.1 Permettere ai client di connettersi

Il server dovrebbe usare qualcosa come una 'lista di controllo di accessò (contenuta nel file di configurazione oppure altrove) che venga letta all'accensione ed usata per decidere quali host possano usare i client per connettersi. Sia la dichiarazione 'deny' (negato) che 'allow' (permesso) dovrebbero essere implementate per fornire la flessibilità necessaria per il controllo di accesso degli hosts.

8.12.2 Operatori

Concedere i privilegi di operatore ad una persona che distruttiva, può avere conseguenze gravi per il buon funzionamento della rete IRC, proprio a causa dei poteri che a quella persona vengono attribuiti. Cosi l'acquisizione di questi poteri non dovrebbe essere cosa facile. La configurazione corrente richiede l'uso di due parole d ordine, sebbene unadi esse sia di solito facilmente intuibile. Se si memorizzano le parole d'ordine per gli operatori nei file di configurazione è consigliabile codificarle ed immagazzinarle in un formato criptato (per esempio usando il crypt(3) di Unix) per prevenire facili ruberie.

8.12.3 Permettere ai server di connettersi

l'interconnessione di server non è una cosa banale: una cattiva connessione può avere una grande influenza sull'efficienza di IRC: cosi ogni server dovrebbe avere una lista di server ai quali possa collegarsi, e di server che possono collegarsi con lui. Per nessuna ragione o circostanza un server dovrebbe permettere ad un arbitrario host di connettersi come server. In aggiunta alla lista di server che possono e non possono connettersi, il file di configurazione dovrebbe anche contenere la parola d ordine ed altre caratteristiche di quel link.

8.12.4 Administrivia

Per fornire delle risposte valide ed accurate al comando ADMIN (si veda la sezione 4.3.7), il server dovrebbe trovare i relativi dettagli nel file di configurazione.

8.13 Associazione ai Canali

Il server corrente permette ad ogni utente locale registrato di associarsi ad un numero massimo di 10 canali. Non c'è limite imposto ad utenti non locali, cosi che il server rimane (ragionevolmente) coerente con tutti gli altri per quanto riguarda l'associazione ai canali.

9. PROBLEMI CORRENTI

Vi sono un certo numero di problemi già rilevati su questo protocollo, che si spera possano essere risolti in un prossimo futuro riscrivendo il software. Attualmente sono in corso tentativi per trovare delle soluzioni efficienti per questi problemi.

9.1 Adattamento a situazioni di dimensioni rilevanti.

É riconosciuto ampiamente che questo protocollo non si adatta sufficientemente bene quando opera su larga scala. Il maggior problema viene dalla necessità che tutti i server sappiano di tutti gli altri server ed utenti e che le informazioni che li riguardano siano aggiornate non appena subiscono cambiamenti. É inoltre desiderabile tenere basso il numero dei servers, così che la lunghezza del percorso tra due punti sia la più breve [Ndr - Evidentemente in termini di numero di nodi intermedi] e che l'albero della rete il più ramificato possibile.

9.2 Etichette
l'attuale protocollo IRC prevede tre tipi di etichette: il nick, il none del canale ed il nome del server. Ognuno di questi tipi ha il suo ambito di validità specifico e non sono permessi duplicati all'interno di quell'ambito. Attualmente è possibile per gli utenti scegliere un'etichetta in ognuno dei tre ambiti, con la possibilità di creare collisioni.

è chiaro a molti che questo problema pone la necessità di una revisione, con un piano di nomi unici che non collidano, per canali e nicks. Altrettanto desiderabile appare una soluzione che permetta un organizzazione ad albero ciclico per la rete.

9.2.1 Nicks
L'idea dei nick su IRC è molto conveniente da usare per gli utenti che parlano gli uni con gli altri fuori da un canale, ma c'è solamente uno spazio finito per i nick come ci si poteva tranquillamente aspettare, non è inconsueto che più persone vogliano usare lo stesso nick. Se un medesimo nickname viene scelto da due persone che usano questo protocollo, o una delle due non riuscirà nel suo intento, oppure tutte e due saranno rimosse con l'uso di KILL (4.6.1).

9.2.2 Canali
l'attuale organizzazione dei canali richiede che tutti i server siano a conoscenza di tutti i canali, dei loro membri e delle loro proprietà. Otre al non perfetto adattamento alle varie condizioni dimensionali della rete, anche la questione della privacy costituisce un problema preoccupante. Una collisone di canali viene trattata come un evento inclusivo (entrambe le persone che creano un canale sono considerate membri di esso) piuttosto che come un evento esclusivo, del tipo di quello usato per risolvere la collisione fra nicks.

9.2.3 Servers
Sebbene il numero dei server sia di solito piccolo, se rapportato al numero di utenti e di canali, devono pero essere conosciuti globalmente, o separatamente uno per uno oppure compresi dentro uno schema (mask).

9.3 Algoritmi
In qualche circostanza, nel software per il server, non è stato possibile evitare gli algoritmi di ordine N^2 quali il controllo della lista di canali di un insieme di clients. Nell'attuale versione del server non sono compresi controlli sulla coerenza dei database: ogni server presume che il suo vicino sia corretto. Questo spalanca le porte a molti problemi se un server che si connette ha dei bug oppure introduce contraddizioni nella rete esistente.

A causa della mancanza di garanzia dell'unicità per le etichette interne e globali, si danno molto spesso dei problemi di coordinamento temporale (race condition). Queste condizioni hanno generalmente origine dalfattoche i messaggi impiegano del tempo per percorrere la rete IRC ed avere effetto su di essa. Anche se si cambia scegliendo un sistema unico di etichette, rimane aperto il problema inerente i comandi relativi ai canali che vengono perduti.