RFC1459 (French)

Groupe de Travail Réseau
Requête pour commentaires: 1459

J. Oikarinen
D. Reed
Mai 1993

Protocole de discussions relayées par internet (IRC)

Statut de ce mémo

Ce mémo définit un protocole expérimental pour la communauté internet. Des discussions et suggestions d'améliorations sont attendues. Veuillez vous référer à la version courante de "IAB Official Protocol Standards" pour connaître l'état de standardisation et le statut de ce protocole. La distribution de ce mémo n'est pas limitée.

Résumé

Le protocole IRC a été développé au cours des 4 dernières années. Il a été initialement implémenté pour permettre aux utilisateurs d'un BBS de discuter entre eux. Maintenant, il est utilisé sur un réseau mondial de serveurs et de clients, et a du mal à gérer sa croissance. Au cours des deux dernières années, le nombre moyen d'utilisateurs connectés au réseau IRC principal a été multiplié par 10.

Le protocole IRC est un protocole en mode texte, dont le client le plus simple est n'importe quel programme TCP capable de se connecter à un serveur.

Table des matières

1. Introduction

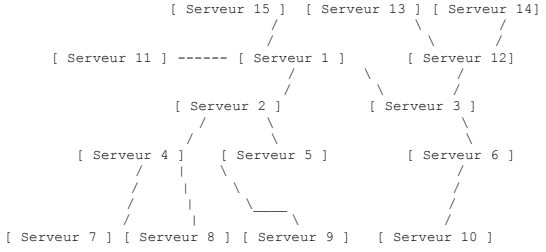
Le protocole IRC (Internet Relay Chat) a été conçu pendant de nombreuses années pour l'usage de conférences en mode texte. Ce document décrit le protocole IRC actuel.

Le protocole IRC a été développé sur des systèmes utilisant le protocole réseau TCP/IP, bien qu'il n'y ait pas de raison que cela reste la seule sphère dans laquelle il opère.

L'IRC, en lui-même, est un système de téléconférence qui (grâce à l'utilisation d'un modèle client/serveur) est adapté à une exécution sur de nombreuses machines, de façon distribuée. Une configuration type comprend un processus unique (le serveur) qui fournit un point d'accès pour les clients (ou d'autres serveurs), et qui traite l'acheminement / le multiplexage requis de messages, ainsi que d'autres fonctions.

1.1 Serveurs

Le serveur est la colonne vertébrale de l'IRC. Il fournit un point auquel les clients peuvent se connecter pour parler entre eux, et un point auquel les autres serveurs peuvent se connecter, formant un réseau IRC. La seule configuration de réseau autorisée est celle d'un arbre [voir Fig. 1] où chaque serveur agit comme un noeud central pour la partie du réseau qu'il voit.



[Fig. 1. Format d'un réseau de serveur IRC]

1.2 Clients

Un client est tout ce qui se connecte à un serveur et qui n'est pas un autre serveur. Chaque client est différencié des autres clients par un pseudonyme unique ayant une longueur maximale de neuf (9) caractères. Voir dans les règles de grammaire du protocole ce qui est autorisé et ce qui ne l'est pas dans un pseudonyme. En plus du pseudonyme, tous les serveurs doivent connaître les informations suivantes sur tous les clients : le vrai nom de l'hôte sur lequel le client est exécuté, le nom de l'utilisateur du client sur cet hôte, et le serveur auquel le client est connecté.

1.2.1 Opérateurs

Pour permettre de maintenir un niveau d'ordre raisonnable dans un réseau IRC, une catégorie de clients spéciale (les opérateurs) est autorisée à exécuter des fonctions de maintenance générale sur le réseau. Bien que les pouvoirs donnés aux opérateurs puissent être considérés comme 'dangereux', ils sont néanmoins indispensables. Les opérateurs doivent être capables de faire certaines tâches de base, telles que la déconnexion et la reconnexion de serveurs, ce qui est nécessaire pour prévenir les problèmes à long terme de mauvais routage réseau. Étant donnée cette nécessité, le protocole décrit ici n'autorise que les opérateurs à effectuer ces fonctions. Voir les section 4.1.7 (SQUIT) et 4.3.5 (CONNECT).

Un pouvoir plus controversé des opérateurs est la possibilité de retirer par la force un utilisateur connecté au réseau, c'est à dire que les opérateurs peuvent clore une connexion entre un client et un serveur. La justification à cela est délicate puisque son abus est à la fois destructif et agaçant. Pour plus de détails concernant ce type d'actions, voir la section 4.6.1 (KILL).

1.3 Les canaux

Un canal est un groupe nommé d'un ou plusieurs clients qui recevront tous les messages adressés à ce canal. Les canaux sont créés implicitement quand le premier client y accède, et le canal disparaît lorsque le dernier client le quitte. Tant qu'un canal existe, tous les clients peuvent y accéder en utilisant le nom du canal.

Les noms de canaux sont des chaînes de caractères (commençant par un caractère '&' ou '#') d'une longueur maximale de 200 caractères. En dehors du fait que le premier caractère doive être un '&' ou un '#', la seule restriction sur le nom d'un canal est qu'il ne peut pas contenir d'espace (' '), de contrôle G (^G ou ASCII 7), ou de virgule (',' qui est utilisée comme séparateur de liste dans le protocole).

Il y a deux types de canaux autorisés par ce protocole. L'un est un canal distribué, qui est connu de tous les serveurs connectés au réseau. Ces canaux commencent par un '#'. L'autre type de canal, reconnaissable à leur nom qui commence par un '&', est marqué comme n'étant accessible qu'aux clients du serveur où le canal existe. En plus de ces deux types, il existe différents modes de canaux, permettant de modifier leur comportement individuel. Voir la section 4.2.3 (commande MODE) pour avoir plus de détails à ce sujet.

Pour créer un nouveau canal, ou pour faire partie d'un canal existant, un utilisateur doit accéder au canal. Si le canal n'existe pas avant l'accès, le canal est créé et l'utilisateur créateur devient opérateur de canal. Si le canal existait déjà au moment de l'accès, l'autorisation ou non d'accès dépend du mode du canal. Par exemple, si le canal est en "invités uniquement" (+i), vous ne pourrez joindre le canal que si vous êtes invité. Le protocole spécifie qu'un utilisateur peut être membre de plusieurs canaux à la fois, mais une limite de dix (10) canaux est recommandée comme étant amplement suffisante aussi bien pour les utilisateurs novices que pour les experts. Voir la section 8.13 pour plus d'informations à ce sujet.

Si le réseau IRC devient disjoint en raison d'une division entre deux serveurs, le canal, de chaque côté, est composé des clients qui sont connectés aux serveurs du côté respectif de la division, et ils disparaissent de l'autre côté de la division. Lorsque la division est réparée, les serveurs se reconnectant se communiquent entre eux qui, d'après eux, est dans chaque canal, et le mode de ce canal. Si le canal existe des deux côtés, les accès et les modes sont interprétés de façon inclusive pour que les deux côtés de la nouvelle connexion soient d'accord sur quels clients sont dans quels canaux et quels modes ont les canaux.

1.3.1 Les opérateurs de canaux

Les opérateurs de canaux (aussi appelés "chanop") sur un canal donné, sont considérés comme étant propriétaires du "canal". A ce titre, les opérateurs de canaux sont dotés de certains pouvoirs qui leur permettent de garder le contrôle et une forme de salubrité à leur canal. En tant que propriétaire d'un canal, un opérateur de canal n'est pas tenu d'avoir de raisons pour agir, toutefois si leurs actions sont généralement antisociales ou abusives, il pourrait être raisonable de demander à un opérateur IRC d'intervenir, ou pour les utilisateurs de simplement partir et aller ailleurs former leur propre canal.

Les commandes réservées aux opérateurs de canaux sont :
KICK - Éjecte un client d'un canal

MODE - Change le mode d'un canal
INVITE - Invite un client dans un canal à accès sur invitation (mode +i)
TOPIC - Change le titre du canal, dans un canal en mode +t

Un opérateur de canal est identifié par un symbole '@' devant son pseudonyme à chaque fois qu'il est utilisé en association avec le canal (c'est à dire lors des réponses aux commandes NAMES, WHO et WHOIS)

2. La spécification IRC

2.1 Aperçu

Le protocole décrit ici vaut aussi bien pour les connexions des serveurs que pour celles des clients. Néanmoins, il y a plus de restrictions sur les connexions des clients (qui sont considérées comme plus douteuses) que sur les connexions des serveurs.

2.2 Les jeux de caractères

Aucun jeu de caractères n'est imposé. Le protocole est basé sur un jeu de caractères de huit (8) bits, qui forment un octet ; cependant, certains codes sont utilisés en tant que codes de contrôle, et agissent comme délimiteurs de messages.

Indépendamment du fait qu'il s'agisse d'un protocole 8 bits, les délimiteurs et les mots-clés sont tels que le protocole est utilisable d'un terminal US-ASCII et d'une connexion telnet.

Étant donnée l'origine scandinave de l'IRC, les caractères {}| sont considérés comme étant respectivement les équivalents en minuscules des caractères []\. Ceci est particulièrement important pour déterminer l'équivalence de deux pseudonymes.

2.3 Messages

Les serveurs et les clients s'envoient chacun des messages qui peuvent ou non générer une réponse. Si le message contient une commande valide, comme il est décrit dans les sections suivantes, le client devrait s'attendre à une réponse comme spécifié, mais il n'est pas conseillé d'attendre éternellement une réponse. La communication de client à serveur, et de serveur à serveur est essentiellement de nature asynchrone.

Chaque message IRC peut contenir jusqu'à trois parties : le préfixe (optionnel), la commande, et les paramètres de la commande (il peut y en avoir jusqu'à 15). Le préfixe, la commande, et tous les paramètres sont séparés par un (ou plusieurs) caractère(s) ASCII espace (0x20).

La présence d'un préfixe est indiquée par un simple caractère ASCII deux-points (':', 0x3b), qui doit être le premier caractère du message. Il ne doit pas y avoir de trou (d'espace blanc) entre les deux-points et le préfixe. Le préfixe est utilisé pour indiquer la véritable origine du message. S'il n'y a pas de préfixe, le message est considéré comme ayant pour origine la connexion de laquelle il est issu. Les clients ne doivent pas utiliser de préfixe lorsqu'ils envoient un message d'eux-mêmes. S'il utilise un préfixe, le seul valable est le pseudonyme associé au client. Si la source identifiée par le préfixe n'est pas trouvée dans la base de données interne du serveur, ou si la source est enregistrée avec une liaison différente de celle avec laquelle le message est arrivé, le serveur doit ignorer le message en silence.

La commande doit être soit une commande IRC valide, soit un nombre de trois (3) chiffres représentés en texte ASCII.

Les messages IRC sont toujours des lignes de caractères terminés par une paire CR-LF (retour chariot - saut de ligne), et ces messages ne doivent pas dépasser 512 caractères de long, en comptant tous les caractères y compris le CR-LF final. Donc, il y a un maximum de 510 caractères autorisés pour la commande et ses paramètres. Il n'y a pas de système permettant une ligne de continuation de message. Voir la section 7 pour les implémentations actuelles.

2.3.1 Le format de message en 'pseudo' BNF

Les messages du protocole doivent être extraits du flux continu d'octets. La solution actuelle consiste à désigner deux caractères donnés, CR et LF, comme séparateurs de messages. Les messages vides sont ignorés en silence, ce qui permet l'usage d'une suite de CR-LF entre les messages sans problèmes supplémentaires.

Le message extrait est décomposé en un <préfixe>, <commande> et liste de paramètres correspondant soit au composant <milieu> ou <fin>.

La représentation BNF de ceci est :
<message> ::= [':' <préfixe> <espace>] <command> <params> <crlf>
<préfixe> ::= <nom de serveur> | <pseudo> ['!' <utilisateur>] ['@' <hôte>]
<command> ::= <lettre> { <lettre> } | <nombre> <nombre> <nombre>
<espace> ::= ' ' { ' ' }
<params> ::= <espace> [':' <fin> | <milieu> <params>]

<milieu> ::= <Toute séquence **non vide** d'octets à l'exclusion de ESPACE, NUL, CR, LF, le premier d'entre eux étant différent de ':'>
<fin> ::= <Toute suite, éventuellement vide, d'octets, à l'exception de NUL, CR et LF>
<crlf> ::= CR LF

NOTES:
1) <espace> est constitué uniquement de caractère(s) ESPACE (0x20). Notez particulièrement que la TABULATION et tout autre caractère de contrôle sont considérés comme ESPACE-NON-BLANC.

2) Après avoir extrait la liste de paramètres, tous les paramètres sont égaux, et correspondent soit à <milieu> soit à <fin>. <Fin> est une simple astuce syntaxique pour autoriser ESPACE dans le paramètre.

3) Le fait que CR et LF ne puissent pas apparaître dans la chaîne paramètre est une simple assertion. Cela pourrait changer dans le futur.

4) Le caractère NUL n'est pas spécial dans la construction du message, et pourrait a priori être à l'intérieur d'un message, mais cela complexifierait la gestion ordinaire des chaînes en C. C'est pourquoi NUL n'est pas autorisé dans les messages.

5) Le dernier paramètre peut être une chaîne vide.

6) L'utilisation d'un préfixe étendu (['!' <utilisateur>] ['@' <hôte>]) ne doit pas être utilisé dans la communication entre serveurs, et est destiné uniquement à la communication serveur vers client, dans le but de fournir au client des informations utiles à propos de l'origine du message sans nécessiter de requêtes supplémentaires.

La plupart des messages du protocole spécifient une sémantique additionnelle, et la syntaxe pour les chaînes de paramètres extraites est dictée par leur position dans la liste. Par exemple, de nombreuses commandes de serveurs vont considérer que le premier paramètre après la commande est la liste de destinataires, ce qui peut être décrit avec :
<destinataire> ::= <à> ["," < destinataire >]
<à> ::= <canal> | < utilisateur > '@' <nom de serveur> | <pseudo> | <masque>
<canal> ::= ('#' | '&') <chaîne canal>
<nom de serveur> ::= <hôte>
<hôte> ::= voir RFC 952 [DNS:4] pour les détails sur les noms d'hôte autorisés
<pseudo> ::= <lettre> { <lettre> | <nombre> | <spécial> }
<masque> ::= ('#' | '$') <chaîne canal>
<chaîne canal> ::= <n'importe quel code 8bits excepté ESPACE, BELL, NUL, CR, LF et virgule (,)>

Les autres paramètres sont :
<utilisateur> ::= <non blanc> { <non blanc> }
<lettre> ::= 'a' ... 'z' | 'A' ... 'Z'
<nombre> ::= '0' ... '9'
<spécial> ::= '-' | '[' | ']' | '\' | '`' | '^' | '{' | '}'

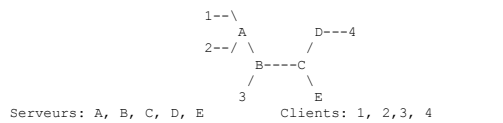
<non blanc> ::= <n'importe quel code 8bits excepté ESPACE (0x20), NUL(0x0), CR(0xd), et LF(0xa) >

2.4 Réponses numériques

La plupart des messages envoyés aux serveurs génèrent une réponse. Les réponses les plus courantes sont des réponses numériques, aussi bien pour les messages d'erreurs que pour les réponses normales. La réponse numérique est envoyée comme un message contenant le préfixe de l'expéditeur, le nombre de trois chiffres, et le destinataire de la réponse. Une réponse numérique ne peut pas émaner d'un client, et tout message de ce style reçu par un serveur doit être ignoré silencieusement. Hormis cela, une réponse numérique est un message comme un autre, si ce n'est que le mot-clé est composé de trois chiffres, plutôt que d'une suite de lettre. Une liste des réponses possibles est fournie à la section 6.

3. Concepts IRC

Cette section décrit les concepts sous-jacents à l'organisation du protocole IRC et comment les implémentations actuelles délivrent les différentes classes de messages.



[Fig. 2. Exemple d'un petit réseau IRC]

3.1 Communication un à un

La communication sur une base un à un n'est utilisée que par les clients, étant donné que la plupart du trafic serveur/serveur n'est pas dû aux serveurs qui se parlent entre eux. Afin de fournir un moyen sécurisé pour les clients de parler entre eux, il est nécessaire que tous les serveurs, pour atteindre un client, soient capables d'envoyer un message dans une seule direction sur l'arbre des connexions. Le chemin d'un message remis est le plus court entre deux points sur l'arbre.

Les exemples suivants se réfèrent tous à la figure 2 ci-dessus.

Exemple 1 :
Un message entre les clients 1et 2 n'est vu que par le serveur A, qui l'envoie directement au client 2.
Exemple 2 :
Un message entre les clients 1 et 3 est vu par les serveurs A & B, et par le client 3. Aucun autre client n'est autorisé à voir le message.
Exemple 3 :
Un message entre les clients 2 et 4 n'est vu que par les serveurs A, B, C & D et par le client 4.

3.2 Un à plusieurs

Le but principal d'IRC est de fournir un forum qui permette de réaliser des conférences de façon simple et efficace (conversation un à plusieurs). L'IRC offre plusieurs moyens de le réaliser, chacun avec des buts différents.

3.2.1 À une liste

Le moyen le moins efficace d'avoir une conversation un à plusieurs consiste, pour chaque client, à parler à une 'liste' d'utilisateurs. La façon dont cela se fait est triviale : chaque client donne une liste de destinataires auxquels le message doit être délivré, et le serveur disperse le message et en distribue une copie à chacun des destinataires. Ce n'est pas aussi efficace que l'utilisation d'un groupe puisque la liste de destinataires est décomposée et la distribution a lieu sans vérifier si le message est envoyé en double sur un même chemin.

3.2.2 À un groupe (canal)

Sur IRC, un canal a un rôle équivalent à celui d'un groupe de multi-diffusion ; son existence est dynamique (il va et vient au fur et à mesure que les gens accèdent et quittent le canal) et la conversation qui a lieu sur un canal n'est envoyée qu'aux serveurs qui ont des utilisateurs sur ce canal. Cette action est alors répétée par chaque combinaison client/serveur jusqu'à ce que le message original ait atteint tous les membres d'un canal.

Les exemples suivants se réfèrent tous à la figure 2.

Exemple 4 :
Pour tout canal qui contient un seul client, les messages du canal vont au serveur et nulle part ailleurs.
Exemple 5 :
Il y a deux clients dans un canal. Tous les messages traversent le chemin comme s'ils étaient des messages privés entre les deux clients en dehors du canal.
Exemple 6 :
Les clients 1, 2 et 3 sont dans un canal. Tous les messages adressés à un canal sont envoyés à tous les clients, et à ceux des serveurs qui seraient traversés par le message s'il était un message privé entre deux clients. Si le client 1 envoie un message, il est envoyé au client 2, et par le serveur B, au client 3.

3.2.3 À un masque d'hôte/de serveur

Afin de fournir aux opérateurs IRC un mécanisme pour envoyer des messages à un grand nombre d'utilisateurs apparentés, on fournit des masques de serveurs et d'hôtes. Ces messages sont envoyés aux serveurs et aux hôtes dont l'adresse correspond au masque. Ces messages ne sont envoyés qu'aux endroits où il y a des utilisateurs, de la même façon que pour les canaux.

3.3 Un à tous

Les messages un à tous peuvent être décrits comme des messages de diffusion, envoyés à tous les clients, tous les serveurs, ou les deux. Sur un grand réseau d'utilisateurs et de serveurs, un simple message peut générer beaucoup de trafic, puisqu'il est envoyé à travers le réseau pour atteindre toutes les destinations.

Pour certains messages, il est nécessaire de les diffuser à tous les serveurs, afin que les informations de statut de chaque serveur soient raisonnablement identiques entre tous les serveurs.

3.3.1 Client à client

Il n'y a pas de classe de message qui, à partir d'un simple message, résulte en un message envoyé à tous les autres clients.

3.3.2 Client à serveur

La plupart des commandes qui résultent en un changement d'état (tels que l'appartenance à un canal, le mode d'un canal, le statut d'un utilisateur, etc.) doivent être, par défaut, envoyés à tous les serveurs, et le client ne peut pas changer cette distribution.

3.3.3 Serveur à serveur

Bien que la plupart des messages entre les serveurs soient distribués à 'tous' les autres serveurs, cela n'est nécessaire que pour les messages qui affectent soit un utilisateur, soit un canal, soit un serveur. Etant donné que cela constitue l'essentiel des éléments de l'IRC, la quasi-totalité des messages issus d'un serveur est diffusée à tous les autres serveurs connectés.

4. Détails des messages

Les pages suivantes décrivent les messages reconnus par les serveurs IRC et les clients. Toutes les commandes décrites dans cette section doivent être implémentées par tous les serveurs utilisant ce protocole.

Si la réponse ERR_NOSUCHSERVER est reçue, cela signifie que le paramètre <serveur> n'a pas été trouvé. Le serveur ne doit alors plus envoyer d'autres réponses pour cette commande.

Le serveur auquel un client est connecté doit traiter le message complet, et retourner les messages d'erreur appropriés. Si le serveur rencontre une erreur fatale en décomposant un message, une erreur doit être envoyée au client et la décomposition interrompue. Peuvent être considérés comme une erreur fatale une commande incorrecte, une destination inconnue du serveur (noms de serveur, pseudo, et noms de canaux entrent dans cette catégorie), un nombre de paramètres insuffisant, ou un niveau de privilège insuffisant.

Si un jeu de paramètres complet est présent, la validité de chacun d'entre eux doit être vérifiée, et les réponses appropriées envoyées au client. Dans le cas de messages dont la liste de paramètres utilise une virgule comme séparateur, une réponse doit être envoyée à chaque élément.

Dans les exemples suivants, certains messages apparaissent au format complet :
:Nom COMMANDE paramètre liste

Ces exemples représentent un message de "Nom" dans le transfert entre serveurs, où il est essentiel d'inclure le nom de l'expéditeur originel du message, de façon à ce que les serveurs distants puissent renvoyer un message le long d'un chemin valide.

4.1 Établissement de connexion

Les commandes décrites dans cette section sont utilisées pour établir une connexion avec un serveur IRC, aussi bien par un client que par un serveur, ainsi qu'une déconnexion correcte.

Une commande "PASS" n'est pas nécessaire pour établir une connexion, mais elle doit précéder la combinaison suivante des messages NICK/USER. Il est fortement recommandé que toutes les connexions de serveurs utilisent un mot de passe afin de donner un niveau de sécurité satisfaisant aux connexions. L'ordre des commandes recommandé pour l'enregistrement d'un client est le suivant :
1. Message PASS
2. Message NICK
3. Message USER

4.1.1 Message PASS

Commande : PASS
Paramètres : <mot de passe>

La commande PASS est utilisée pour définir le 'mot de passe de connexion'. Le mot de passe peut et doit être défini avant toute tentative de connexion. A l'heure actuelle, cela signifie que les clients doivent envoyer une commande PASS avant d'envoyer la combinaison NICK/USER, et que les serveurs **doivent** envoyer une commande PASS avant toute commande SERVER. Le mot de passe fourni doit correspondre à celui contenu dans les lignes C/N (pour les serveurs) ou dans les lignes I (pour les clients). Il est possible d'envoyer plusieurs commandes PASS avant de s'enregistrer, mais seule la dernière est utilisée pour la vérification, et elle ne peut plus changer une fois la connexion établie.

Réponses numériques :

```
ERR_NEEDMOREPARAMS              ERR_ALREADYREGISTRED
```

Exemple:
PASS motdepasssecretici

4.1.2 Message NICK

Commande : NICK
Paramètres : <pseudonyme> [<compteur de distance>]

Le message NICK est utilisé pour donner un pseudonyme à un utilisateur, ou pour changer le pseudonyme précédent. Le paramètre <compteur de distance> n'est utilisé que par les serveurs, et sert à indiquer quelle est la distance entre un utilisateur et son serveur local. Une connexion locale a un compteur de distance de zéro. Si un client fournit un compteur de distance, il doit être ignoré.

Si un message NICK arrive à un serveur qui connaît déjà un autre client de pseudo identique, une collision de pseudonymes a lieu. Le résultat d'une collision de pseudonymes est la suppression de toutes les instances du pseudonyme de la base du serveur, et l'envoi d'un message KILL afin de retirer le pseudonyme des bases de données de tous les autres serveurs. Si le message NICK à l'origine de la collision de pseudonymes est un changement de pseudonyme, alors le pseudo originel (l'ancien) doit aussi être retiré.

Si le serveur reçoit un NICK identique d'un client auquel il est connecté directement, il peut envoyer un ERR_NICKCOLLISION au client local, ignorer la commande NICK, et ne pas générer de KILLs.

Réponses numériques :

```
ERR_NONICKNAMEGIVEN             ERR_ERRONEUSNICKNAME
ERR_NICKNAMEINUSE               ERR_NICKCOLLISION
```

Exemples:

`NICK Wiz` ; Ajout d'un nouveau pseudo "Wiz".
`:WiZ NICK Kilroy` ; WiZ change son pseudo en Kilroy.

4.1.3 Message USER

Commande: USER
Paramètres: <nom d'utilisateur> <hôte> <nom de serveur> <nom réel>

Le message USER est utilisé au début d'une connexion pour spécifier le nom d'utilisateur, le nom d'hôte, le nom de serveur, et le véritable nom d'un nouvel utilisateur. Il est aussi utilisé lors de la communication entre les serveurs pour indiquer l'arrivée d'un nouvel utilisateur sur IRC, puisque c'est seulement après avoir envoyé et le USER et le NICK qu'un utilisateur devient enregistré.

Entre serveurs, USER doit être préfixé du pseudonyme du client. Notez que le nom d'hôte et le nom de serveur sont généralement ignorés par le serveur IRC quand la commande USER vient directement d'un client (pour des raisons de sécurité), mais sont utilisés dans la communication de serveur à serveur. Cela signifie que NICK doit toujours être envoyé à un serveur distant quand un nouvel utilisateur est ajouté au reste du réseau, avant que le USER correspondant soit envoyé.

Notez aussi que le paramètre 'vrai nom' doit être le dernier paramètre, car il peut contenir des espaces, et il doit être préfixé par deux points (':') de façon à être reconnu comme tel.

Puisqu'il est facile pour un client de mentir sur son nom si on se base uniquement sur le message USER, il est recommandé d'utiliser un "serveur d'identité". Si l'hôte dont l'utilisateur se connecte a un serveur de ce type activé, le nom d'utilisateur est défini par la réponse de ce "serveur d'identité".

Réponses numériques :

```
ERR_NEEDMOREPARAMS              ERR_ALREADYREGISTRED
```

Exemples:

`USER guest tolmoon tolsun :Ronnie Reagan` ; Utilisateur s'enregistrant avec un nom d'utilisateur de "guest" et un vrai nom de "Ronnie Reagan". `:testnick USER guest tolmoon tolsun :Ronnie Reagan` ; message entre les serveurs contenant le pseudonyme à qui appartient la commande USER.

4.1.4 Message SERVER

Commande: SERVER
Paramètres: <nom de serveur> <compteur de distance> <info>

Le message SERVER est utilisé pour dire à un serveur que l'autre bout de la connexion est un autre serveur. Ce message est aussi utilisé pour transmettre les données du serveur à tout le réseau. Quand un nouveau serveur se connecte au réseau, les informations le concernant sont diffusées à tout le réseau. <Compteur de distance> est utilisé pour donner à chaque serveur des informations sur leurs distances aux différents serveurs. Avec la liste complète des serveurs, il serait possible de construire une carte complète de l'arbre des serveurs, mais les masques d'hôtes l'en empêchent.

Le message SERVER doit être accepté uniquement (a) soit d'une connexion qui n'est pas encore enregistrée et qui essaie de s'enregistrer en tant que serveur, ou (b) d'une connexion existante d'un autre serveur, pour introduire un nouveau serveur au delà de ce serveur.

La plupart des erreurs qui ont lieu à la réception d'une commande SERVER résulte en la coupure de la connexion avec l'hôte destinataire (serveur destinataire). Les réponses d'erreur sont généralement envoyées en utilisant la commande "ERROR" plutôt qu'une réponse numérique, car la commande ERROR a plusieurs propriétés qui la rendent utile dans ce cas.

Si un message SERVER est traité et tente d'introduire un serveur déjà connu du serveur receveur, la connexion avec ce serveur doit être coupée (en suivant les procédures correctes), car une route dupliquée s'est formée avec un serveur, et la nature acyclique de l'arbre IRC est rompue.

Réponses numériques :

```
ERR_ALREADYREGISTRED
```

Exemples :

```
SERVER test.oulu.fi 1 :[tolsun.oulu.fi] Experimental server
```
; Le nouveau serveur test.oulu.fi se présente et tente de s'enregistrer. Le nom d'hôte est test.oulu.fi.

```
:tolsun.oulu.fi SERVER csd.bu.edu 5 :BU Central Server
```
; Le serveur tolsun.oulu.fi est notre lien vers csd.bu.edu qui est à 5 nœuds de nous.

4.1.5 Message OPER

Commande: OPER
Paramètres: <utilisateur> <mot de passe>

Le message OPER est utilisé par un utilisateur normal pour obtenir le privilège d'opérateur. La combinaison de <utilisateur> et <mot de passe> est nécessaire pour obtenir le privilège Opérateur.

Si le client envoyant la commande OPER fournit un mot de passe correct pour l'utilisateur donné, le serveur informe le reste du réseau du nouvel opérateur en envoyant un "MODE +o" pour le pseudonyme.

Le message OPER n'a lieu qu'entre un client et un serveur.

Réponses numériques :

```
ERR_NEEDMOREPARAMS              RPL_YOUREOPER
ERR_NOOPERHOST                  ERR_PASSWDMISMATCH
```

Exemple:

```
OPER foo bar
```
; Tentative d'enregistrement en tant qu'opérateur, de l'utilisateur "foo" utilisant "bar" comme mot de passe.

4.1.6 Message QUIT

Commande: QUIT
Paramètres: [<Message de départ >]

Une session de client se termine par un message QUIT. Le serveur doit rompre la connexion avec le client qui envoie un message QUIT. Si un <Message de départ> est fourni, il sera transmis au lieu du message par défaut, le pseudonyme.

Lorsqu'une division réseau a lieu (déconnexion de deux serveurs), le message de départ est composé du nom des deux serveurs en cause, séparés par un espace. Le premier nom est celui du serveur toujours connecté, et le second, celui qui est désormais inaccessible.

Si pour une autre raison, une connexion d'un client est fermée sans que le client ait envoyé de message QUIT (par exemple, le programme client se termine, et une fin de fichier est envoyée sur la socket), le serveur doit remplir le message QUIT avec un message reflétant la nature de l'événement à l'origine de cette déconnexion.

Réponses numériques:

```
Aucune.
```

Exemples:

```
QUIT :Parti déjeuner
```
; Format de message préféré.

4.1.7 Message SQUIT

Commande: SQUIT
Paramètres: <serveur> <commentaire>

Le message SQUIT est nécessaire pour signaler le départ ou la mort de serveurs. Si un serveur souhaite rompre une connexion à un autre serveur, il doit envoyer un message SQUIT à ce serveur, en utilisant le nom de ce serveur comme paramètre <serveur>, ce qui clôt la connexion avec le serveur quittant le réseau.

Cette commande est aussi accessible aux opérateurs pour garder un réseau de serveurs IRC connectés proprement. Les opérateurs peuvent également émettre un message SQUIT pour une connexion distante d'un serveur. Dans ce cas, le message SQUIT doit être traité par tous les serveurs entre l'opérateur et le serveur distant, tout en mettant à jour la vue du réseau de chaque serveur comme décrit plus loin.

Le <commentaire> doit être fourni par tous les opérateurs qui exécutent un SQUIT sur un serveur distant (qui n'est pas connecté au serveur sur lequel ils sont actuellement). Le <commentaire> est également rempli par les serveurs qui peuvent y placer un message d'erreur ou autre.

Les deux serveurs, de chaque côté de la connexion qui va être coupée, doivent envoyer un message SQUIT (à tous les serveurs auxquels ils sont connectés) vers tous les serveurs qui sont situés au-delà de ce lien.

De même, un message QUIT doit être envoyé aux autres serveurs du reste du réseau de la part de tous les clients au-delà de ce lien. De plus, tous les membres d'un canal qui perd un membre à cause d'une division réseau doivent recevoir un message QUIT.

Si une connexion à un serveur est terminée prématurément, (par exemple si le serveur à l'extrémité de la liaison meurt), le serveur qui détecte cette déconnexion doit informer le reste du réseau que cette connexion est fermée, et remplir le champ <commentaire> de façon appropriée.

Réponses numériques :

```
ERR_NOPRIVILEGES                ERR_NOSUCHSERVER
```

Exemples:

```
SQUIT tolsun.oulu.fi :Bad Link ?
```
; la liaison au serveur tolson.oulu.fi s'est terminée en raison d'un mauvais lien.

```
:Trillian SQUIT cm22.eng.umd.edu :Server out of control
```
; message de Trillian pour déconnecter "cm22.eng.umd.edu" du réseau en raison d'un "serveur incontrôlable".

4.2 Opérations sur les canaux

Ce groupe de messages s'intéresse à la manipulation de canaux, à leurs propriétés (mode des canaux), et à leur contenu (typiquement des clients). En implémentant ces commandes, la résolution de conflits entre les clients est inévitable, par exemple lorsque deux clients à deux extrémités du réseau envoient des commandes incompatibles. Il est aussi nécessaire aux serveurs de garder l'historique d'un pseudonyme afin que, quand un paramètre <pseudo> est donné, le serveur puisse vérifier dans l'historique si le pseudo n'a pas changé récemment.

4.2.1 Message JOIN

Commande: JOIN
Paramètres: <canal>{,<canal>} [<clé>{,<clé>}]

La commande JOIN est utilisée par un client pour commencer à écouter un canal spécifique. L'accès à un canal est autorisé ou non uniquement par le serveur auquel le client est connecté ; tous les autres serveurs ajoutent automatiquement l'utilisateur au canal quand la demande vient d'un autre serveur. Les conditions qui affectent ceci sont les suivantes :
1. L'utilisateur doit être invité si le canal est en mode "sur invitation seulement"
2. Le pseudo/nom d'utilisateur/nom d'hôte ne doit pas correspondre à un bannissement actif.
3. La bonne clé (mot de passe) doit être fournie si elle est définie.

Ceci est discuté plus en détails dans la description de la commande MODE (voir la section 4.2.3 pour plus de détails).

Une fois qu'un utilisateur a accès à un canal, il reçoit des notifications de toutes les commandes que son serveur reçoit et qui affectent le canal. Cela inclut MODE, KICK, PART, QUIT, et bien sûr PRIVMSG/NOTICE. La commande JOIN doit être diffusée à tous les serveurs du réseau pour qu'ils sachent où trouver qui est dans chaque canal. Cela permet une distribution optimale des messages PRIVMSG/NOTICE du canal.

Si un JOIN a lieu avec succès, on envoie à l'utilisateur le sujet du canal (en utilisant RPL_TOPIC) et la liste des utilisateurs du canal (en utilisant RPL_NAMREPLY), y compris lui-même.

Réponses numériques :

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

Exemples:

`JOIN #foobar` ; accède au canal #foobar.

`JOIN &foo fubar` ; accède au canal &foo en utilisant la clé "fubar".

`JOIN #foo,&bar fubar` ; accède au canal #foo en utilisant la clé "fubar", et &bar en n'utilisant pas de clé.

`JOIN #foo,#bar fubar,foobar` ; accède au canal #foo en utilisant la clé "fubar", et au canal #bar en utilisant la clé "foobar".

`JOIN #foo,#bar` ; accède au canaux #foo and #bar.

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

4.2.2 Message PART

Commande: PART
Paramètres: <canal>{,< canal >}

Le message PART provoque le retrait du client expéditeur de la liste des utilisateurs actifs pour tous les canaux listés dans la chaîne de paramètres.

Réponses numériques:

```
ERR_NEEDMOREPARAMS              ERR_NOSUCHCHANNEL
ERR_NOTONCHANNEL
```

Exemples:

`PART #twilight_zone` ; quitte le canal "#twilight_zone"

`PART #oz-ops,&group5` ; quitte les canaux "&group5" et "#oz-ops".

4.2.3 Message MODE

Commande: MODE

La commande MODE est une commande à utilisation duale sur IRC. Elle permet aussi bien de changer les modes des utilisateurs que ceux des canaux. La raison à ce choix est qu'un jour les pseudonymes deviendront obsolètes et la propriété équivalente sera le canal.

Lors du traitement des messages MODE, il est recommandé de commencer par décomposer le message en entier, puis de réaliser les modifications résultantes.

4.2.3.1 Les modes des canaux

Paramètres: <canal> {[+|-]|o|p|s|i|t|n|b|v} [<limite>] [<utilisateur>] [<masque de bannissement >]

La commande MODE permet aux opérateurs de canal de changer les caractéristiques de 'leur' canal. Les serveurs doivent aussi pouvoir changer les modes du canal, de façon à pouvoir créer des opérateurs.

Les modes disponibles pour les canaux sont les suivants :
o - donne/retire les privilèges d'opérateur de canal
p - drapeau de canal privé
s - drapeau de canal secret
i - drapeau de canal accessible uniquement sur invitation
t - drapeau de sujet de canal modifiable uniquement par les opérateurs
n - pas de messages dans un canal provenant de clients à l'extérieur du canal
m - canal modéré
l - définit le nombre maximal de personnes dans un canal
b - définit un masque de bannissement pour interdire l'accès à des utilisateurs
v - donne/retire la possibilité de parler dans un canal modéré
k - définit la clé du canal (mot de passe)

Lors de l'utilisation des options 'o' et 'b', le nombre de paramètres est restreint à trois par commande, et ce pour n'importe quelle combinaison de 'o' et de 'b'.

4.2.3.2 Modes des utilisateurs

Paramètres: <pseudonyme> {[+|-]|i|w|s|o}

Les modes utilisateurs sont typiquement des modifications qui affectent la façon dont le client est vu par les autres, ou quels types de messages sont reçus. Une commande MODE n'est acceptée que si l'expéditeur du message et le pseudonyme donné en paramètre sont les mêmes.

Les modes disponibles sont :
i - marque un utilisateur comme invisible ;
s - marque un utilisateur comme recevant les notifications du serveur ;
w - l'utilisateur reçoit les WALLOPs ;
o - drapeau d'opérateur.

D'autres modes seront disponibles plus tard.

Si un utilisateur tente de devenir opérateur en utilisant le drapeau "+o", la tentative doit être ignorée. Par contre, il n'y a pas de restriction à ce que quelqu'un se 'deoppe' (en utilisant "-o").

Réponses numériques :

```
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
```

Exemples:

Utilisation des modes de canal:
`MODE #Finnish +im` ; Rend le canal #Finnish modéré et 'uniquement sur invitation'.
`MODE #Finnish +o Kilroy` ; Donne le privilège de 'chanop' à Kilroy sur le canal #Finnish.
`MODE #Finnish +v Wiz` ; Autorise WiZ à parler sur #Finnish.
`MODE #Fins -s` ; Annule le drapeau 'secret' du canal #Fins.
`MODE #42 +k oulu` ; Définit la clé comme "oulu".
`MODE #eu-opers +l 10` ; Définit le nombre maximal d'utilisateurs dans le canal à 10.
`MODE &oulu +b` ; Liste les masques de bannissement du canal.
`MODE &oulu +b *!*@*` ; Interdit à quiconque de venir sur le canal.
`MODE &oulu +b *!*@*.edu` ; Interdit à tout utilisateur dont le nom d'hôte correspond à *.edu d'accéder au canal.

Utilisation des modes d'utilisateur :
`:MODE WiZ -w` ; supprime la réception des messages WALLOPS pour WiZ.
`:Angel MODE Angel +i` ; Message d'Angel pour se rendre invisible.
`MODE WiZ -o` ; WiZ se 'deoppe' (retire son statut d'opérateur). Le contraire ("MODE WiZ +o") ne doit pas être autorisé car cela court-circuiterait la commande OPER.

4.2.4 Message TOPIC

Commande: TOPIC
Paramètres: <canal> [<sujet>]

Le message TOPIC est utilisé pour modifier ou voir le sujet d'un canal. Le sujet du canal <canal> est renvoyé s'il n'y a pas de <sujet> fourni en paramètre. Si le paramètre <sujet> est présent, le sujet du canal changera si le mode du canal le permet.

Réponses numériques :

```
ERR_NEEDMOREPARAMS              ERR_NOTONCHANNEL
RPL_NOTOPIC                     RPL_TOPIC
ERR_CHANOPRIVSNEEDED
```

Exemples:

`:Wiz TOPIC #test :New topic` ; L'utilisateur Wiz définit le sujet.
`TOPIC #test :another topic` ; Change le sujet du canal #test en "another topic".
`TOPIC #test` ; Vérifie le sujet de #test.

4.2.5 Message NAMES

Commande: NAMES
Paramètres: [<canal>{,<canal>}]

En utilisant la commande NAMES, un utilisateur peut obtenir la liste des pseudonymes visibles sur n'importe quel canal qu'il peut voir. Les noms de canaux qu'il peut voir sont ceux qui ne sont ni privés (+p), ni secrets (+s), ou ceux sur lesquels il est actuellement. Le paramètre <canal> spécifie quels sont les canaux dont l'information est voulue, s'ils sont valides. Il n'y a pas de message d'erreur pour les noms de canaux invalides.

Si le paramètre <canal> n'est pas donné, la liste de tous les canaux et de leurs occupants est renvoyée. A la fin de cette liste, sont listés les utilisateurs visibles, mais qui n'appartiennent à aucun canal visible. Ils sont listés comme appartenant au 'canal' "*".

Réponses numériques:

```
RPL_NAMREPLY                    RPL_ENDOFNAMES
```

Exemples:

`NAMES #twilight_zone,#42` ; liste les utilisateurs visibles sur #twilight_zone et #42, si ces canaux vous sont visibles.
`NAMES` ; liste tous les canaux, et tous les utilisateurs visibles.

4.2.6 Message LIST

Commande: LIST
Paramètres: [<canal>{,<canal>} [<serveur>]]

Le message LIST est utilisé pour lister les canaux et leur sujet. Si le paramètre <canal> est utilisé, seul le statut de ces canaux est affiché. Les canaux privés sont listés (sans leur sujet) comme canal "Prv" à moins que le client qui génère la requête soit effectivement sur le canal. De même, les canaux secrets ne sont pas listés du tout, à moins que le client soit un membre du canal en question.

Réponses numériques :

```
ERR_NOSUCHSERVER                RPL_LISTSTART
RPL_LIST                        RPL_LISTEND
```

Exemples:

`LIST` ; Liste tous les canaux.
`LIST #twilight_zone,#42` ; Liste les canaux #twilight_zone et #42

4.2.7 Message INVITE

Commande: INVITE
Paramètres: <pseudonyme> <canal>

Le message INVITE est utilisé pour inviter des utilisateurs dans un canal. Le paramètre <pseudonyme> est le pseudonyme de la personne à inviter dans le canal destination <canal>. Il n'est pas nécessaire que le canal dans lequel la personne est invitée existe, ni même soit valide. Pour inviter une personne dans un canal en mode sur invitation (MODE +i), le client envoyant l'invitation doit être opérateur sur le canal désigné.

Réponses numériques :

```
ERR_NEEDMOREPARAMS              ERR_NOSUCHNICK
ERR_NOTONCHANNEL                ERR_USERONCHANNEL
```

```
ERR_CHANOPRIVSNEEDED
RPL_INVITING                    RPL_AWAY
```

Exemples:

`:Angel INVITE Wiz #Dust` ; L'utilisateur Angel invite WiZ sur le canal #Dust
`INVITE Wiz #Twilight_Zone` ; Commande pour inviter WiZ sur #Twilight_zone

4.2.8 Commande KICK

Commande: KICK
Paramètres: <canal> <utilisateur> [<commentaire>]

La commande KICK est utilisée pour retirer par la force un utilisateur d'un canal (PART forcé).

Seul un opérateur de canal peut kicker un autre utilisateur hors d'un canal. Tout serveur qui reçoit un message KICK vérifie si le message est valide (c'est-à-dire si l'expéditeur est bien un opérateur du canal) avant d'ôter la victime du canal.

Réponses numériques :

```
ERR_NEEDMOREPARAMS              ERR_NOSUCHCHANNEL
ERR_BADCHANMASK                 ERR_CHANOPRIVSNEEDED
ERR_NOTONCHANNEL
```

Exemples:

`KICK &Melbourne Matthew` ; Kick Matthew de &Melbourne
`KICK #Finnish John :Speaking English` ; Kick John de #Finnish en spécifiant "Speaking English" comme raison (commentaire).
`:WiZ KICK #Finnish John` ; Message KICK de WiZ pour retirer John du canal #Finnish

NOTE:

Il est possible d'étendre les paramètres de la commande KICK ainsi :
<canal>{,<canal>} <utilisateur>{,<utilisateur>} [<commentaire>]

4.3 Requêtes et commandes des serveurs

Le groupe de requêtes de commande des serveurs sert à obtenir des informations au sujet de tout serveur connecté au réseau. Tous les serveurs connectés doivent répondre à ces requêtes et répondre correctement. Toute réponse invalide (ou l'absence de réponse) doit être considéré comme un signe de serveur cassé, et il doit se déconnecter / se désactiver au plus tôt, jusqu'à ce que le problème soit résolu.

Dans ces requêtes, lorsqu'il y a un paramètre "<serveur>", cela désigne généralement un pseudonyme, un serveur, ou un joker quelconque. Par contre, chaque paramètre ne doit générer qu'une seule requête et un jeu de réponses.

4.3.1 Message VERSION

Commande: VERSION
Paramètres: [<serveur>]

Le message VERSION est utilisé pour déterminer la version du programme serveur. Un paramètre optionnel <serveur> est utilisé pour obtenir la version d'un programme serveur sur lequel un client n'est pas connecté directement.

Réponses numériques :

```
ERR_NOSUCHSERVER                RPL_VERSION
```

Exemples:

`:Wiz VERSION *.se` ; message de Wiz pour vérifier la version d'un serveur correspondant à "*.se"
`VERSION tolsun.oulu.fi` ; vérifie la version du serveur "tolsun.oulu.fi".

4.3.2 Message STATS

Commande: STATS
Paramètres: [<requête> [<serveur>]]

Le message STATS est utilisé pour obtenir les statistiques d'un serveur. Si le paramètre <serveur> est omis, seule la fin de la réponse STATS est renvoyée. L'implémentation de cette requête dépend énormément du serveur qui répond. Néanmoins, le serveur doit être capable de fournir les informations décrites dans les requêtes ci-dessous (ou équivalent).

Une requête peut être lancée par une unique lettre, qui est vérifiée uniquement par le serveur destination (si le paramètre <serveur> est présent). Elle est transmise aux serveurs intermédiaires, ignorée et inaltérée. Les requêtes suivantes sont celles trouvées dans les implémentations courantes d'IRC, et fournissent une grande partie des informations de configuration du serveur. Bien qu'elles ne soient pas nécessairement gérées de la même façon par d'autres versions, tous les serveurs devraient être capables de fournir une réponse valide à la requête STATS, qui soit compatible avec les formats de réponse actuellement utilisées et le but de ces requêtes.

Les requêtes actuellement gérées sont :
c - renvoie la liste des serveurs qui peuvent se connecter, ou dont les connexions sont acceptées ;
h - renvoie la liste des serveurs qui sont forcés de se comporter comme des feuilles(L) , ou comme des noeuds (H) sur l'arbre des connexions ;
i - renvoie la liste des hôtes dont le serveur accepte les clients ;
k - retourne la liste des combinaisons de noms d'utilisateurs / noms d'hôtes qui sont bannis de ce serveur.
l - renvoie la liste des connexions d'un serveur, et montre, pour chacune d'entre elles, le trafic en octets et en messages, et ce, dans chaque direction ;
m - renvoie la liste des commandes gérées par le serveur, et le nombre d'utilisations de chacune d'entre elles, s'il n'est pas nul ;
o - renvoie la liste des hôtes depuis lesquels un client normal peut devenir opérateur ;
y - montre les lignes Y (Classe) du fichier de configuration du serveur ;
u - renvoie une chaîne décrivant depuis combien de temps le serveur fonctionne.

Réponses numériques :

```
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
```

Exemples:

`STATS m` ; vérifie l'utilisation des commandes pour le serveur sur lequel vous êtes connecté.
`:Wiz STATS c eff.org` ; requête de WiZ pour information sur les lignes C/N du serveur eff.org

4.3.3 Message LINKS

Commande: LINKS
Paramètres: [[<serveur distant>] <masque de serveur >]

Avec LINKS, un utilisateur peut obtenir la liste des serveurs connue d'un serveur. La liste des serveurs doit correspondre au masque, ou s'il n'y a pas de masque, la liste complète des serveurs est renvoyée.

Si le <serveur distant> est fourni, en plus du <masque de serveur>, la commande LINKS est transmise au premier serveur trouvé dont le nom correspond (s'il y en a), et ce serveur doit alors répondre à la requête.

Réponses numériques :

```
ERR_NOSUCHSERVER
RPL_LINKS                       RPL_ENDOFLINKS
```

Exemples:

`LINKS *.au` ; liste tous les serveurs dont le nom correspond à *.au;
`:WiZ LINKS *.bu.edu *.edu` ; message LINKS de WiZ au premier serveur correspondant à *.edu demandant la liste des serveurs correspondant à *.bu.edu.

4.3.4 Message TIME

Commande: TIME
Paramètres: [<serveur>]

Le message TIME est utilisé pour obtenir l'heure locale d'un serveur donné. En absence de paramètre <serveur>, le serveur recevant le message doit répondre à la requête.

Réponses numériques :

```
ERR_NOSUCHSERVER                RPL_TIME
```

Exemples:

`TIME tolsun.oulu.fi` ; demande l'heure du serveur "tolson.oulu.fi"
`Angel TIME *.au` ; L'utilisateur Angel demande l'heure à un serveur correspondant à "*.au"

4.3.5 Message CONNECT

Commande: CONNECT
Paramètres: <serveur destination > [<port> [<serveur distant>]]

Le message CONNECT est utilisé pour forcer un serveur à essayer d'établir immédiatement une nouvelle connexion à un autre serveur. CONNECT est une commande privilégiée et n'est accessible qu'aux opérateurs IRC. Si un serveur distant est précisé, alors ce serveur tente de se connecter au <serveur distant>, sur le <port> donné.

Réponses numériques :

```
ERR_NOSUCHSERVER                ERR_NOPRIVILEGES
ERR_NEEDMOREPARAMS
```

Exemples:

`CONNECT tolsun.oulu.fi` ; Essai de connexion au serveur tolsun.oulu.fi
`:WiZ CONNECT eff.org 6667 csd.bu.edu` ; essai de CONNECT de WiZ pour lier eff.org et csd.bu.edu sur le port 6667.

4.3.6 Message TRACE

Commande: TRACE
Paramètres: [<serveur>]

La commande TRACE est utilisée pour trouver une route vers un serveur donné. Chaque serveur qui traite ce message doit répondre à l'expéditeur en indiquant qu'il est un lien sur le chemin d'acheminement, formant ainsi une chaîne de réponse similaire à celle obtenue par un "traceroute". Après avoir renvoyé sa réponse, il doit ensuite envoyer le message TRACE au serveur suivant, et ce jusqu'à ce que le serveur spécifié soit atteint. Si le paramètre <serveur> est omis, il est recommandé que la commande trace envoie un message à l'expéditeur lui disant à quels serveurs il est directement connecté.

Si la destination spécifiée par <serveur> est en fait un serveur, alors le serveur destinataire doit lister tous les serveurs et les utilisateurs qui y sont connectés. Si la destination spécifiée par <serveur> est en fait un pseudonyme, seule la réponse pour ce pseudonyme est donnée.

Réponses numériques :

```
ERR_NOSUCHSERVER
```

Si le message TRACE est destiné à un autre serveur, tous les serveurs intermédiaires doivent retourner une réponse RPL_TRACELINK pour indiquer que le TRACE est passé par lui et où il va ensuite.

```
RPL_TRACELINK
```

Une réponse TRACE doit être une des réponses numériques suivantes :

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

Exemples:

`TRACE *.oulu.fi` ; TRACE un serveur correspondant à *.oulu.fi
`:WiZ TRACE AngelDust` ; TRACE de WiZ vers le pseudo AngelDust

4.3.7 Commande ADMIN

Commande: ADMIN
Paramètres: [<serveur>]

Le message ADMIN est utilisé pour trouver le nom de l'administrateur d'un serveur donné, ou du serveur courant si le paramètre <serveur> est omis. Tout serveur doit posséder la possibilité de propager les messages ADMIN aux autres serveurs.

Réponses numériques :

```
ERR_NOSUCHSERVER
RPL_ADMINME                     RPL_ADMINLOC1
RPL_ADMINLOC2                   RPL_ADMINEMAIL
```

Exemples:

`ADMIN tolsun.oulu.fi` ; requête ADMIN de tolsun.oulu.fi
`:WiZ ADMIN *.edu` ; requête ADMIN de WiZ pour le premier serveur trouvé qui correspond à *.edu.

4.3.8 Commande INFO

Commande: INFO
Paramètres: [<serveur>]

La commande INFO doit retourner une information qui décrit le serveur : sa version, quand il a été compilé, le numéro de mise à jour, quand il a été démarré, et toute autre information considérée comme pertinente.

Réponses numériques :

```
ERR_NOSUCHSERVER
RPL_INFO                        RPL_ENDOFINFO
```

Exemples:

`INFO csd.bu.edu` ; requête INFO pour csd.bu.edu
`:Avalon INFO *.fi` ; requête INFO d'Avalon à destination du premier serveur trouvé qui correspond à *.fi.
`INFO Angel` ; requête INFO à destination du serveur sur lequel est connecté Angel.

4.4 Envoi de messages

Le but principal du protocole IRC est de fournir une base afin que des clients puissent communiquer entre eux. PRIVMSG et NOTICE sont les seuls messages disponibles qui réalisent effectivement l'acheminement d'un message textuel d'un client à un autre - le reste le rend juste possible et assure que cela se passe de façon fiable et structurée.

4.4.1 Messages privés

Commande: PRIVMSG
Paramètres: <destinataire>{,<destinataire>} <texte à envoyer >

PRIVMSG est utilisé pour envoyer un message privé entre des utilisateurs. <destinataire> est le pseudonyme du destinataire du message. <destinataire> peut aussi être une liste de noms ou de canaux, séparés par des virgules.

Le paramètre <destinataire> peut aussi être un masque d'hôte (masque #) ou un masque de serveur (masque $). Le masque doit contenir au moins un (1) ".", et aucun joker après le dernier ".". Cette limitation a pour but d'empêcher les gens d'envoyer des messages à "#*" ou à "$*", ce qui provoquerait la diffusion à tous les utilisateurs ; l'expérience montre qu'on en abuse plus qu'on en use intelligemment, de façon responsable. Les jokers sont les caractères '*' et '?'. Ces extensions de PRIVMSG ne sont accessibles qu'aux opérateurs.

Réponses numériques:

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

Exemples:

`:Angel PRIVMSG Wiz :Salut, est-ce que tu reçois ce message ?` ; Message d'Angel à Wiz.
`PRIVMSG Angel :oui, je le reçois !` ; Message à Angel.
`PRIVMSG jto@tolsun.oulu.fi :Hello !` ; Message à un client du serveur tolsun.oulu.fi dont le nom est "jto".
`PRIVMSG $*.fi :Server tolsun.oulu.fi rebooting.` ; Message à tous sur les serveurs dont les noms correspondent à *.fi.
`PRIVMSG #*.edu :NSFNet is undergoing work, expect interruptions` ; Message à tous les utilisateurs qui viennent d'un hôte dont le nom correspond à *.edu.

4.4.2 Notice

Commande: NOTICE
Paramètres: <pseudonyme> <texte>

Le message NOTICE s'utilise de la même façon que PRIVMSG. La différence entre NOTICE et PRIVMSG est qu'aucune réponse automatique ne doit être envoyée en réponse à un message NOTICE. Ceci est aussi valable pour les serveurs - ils ne doivent pas renvoyer de message d'erreur à la réception d'un message NOTICE. Le but de cette règle est d'éviter les boucles entre les clients qui enverraient automatiquement quelque chose en réponse à une requête. Cela est typiquement utilisé par des automates (des clients qui ont une intelligence artificielle ou un autre programme interactif contrôlant leurs actions) qui répondent systématiquement aux réponses d'autres automates.

Voir PRIVMSG pour les détails sur les réponses, et pour les exemples.

4.5 Requêtes basées sur les utilisateurs

Les requêtes utilisateurs sont un groupe de commandes dont le but principal est la recherche d'informations sur un utilisateur particulier, ou sur un groupe d'utilisateurs. Lorsqu'on utilise des jokers avec ces commandes, elles ne renvoient les informations que sur les utilisateurs qui vous sont 'visibles'. La visibilité d'un utilisateur est déterminée par la combinaison du mode de cet utilisateur, et des canaux sur lesquels tous les deux êtes.

4.5.1 Requête WHO

Commande: WHO
Paramètres: [<nom> [<o>]]

Le message WHO est utilisé par un client pour générer une requête qui renvoie une liste d'informations qui correspondent au paramètre <nom> donné par le client. Si le paramètre nom est absent, tous les utilisateurs visibles (ceux qui ne sont pas invisibles (mode utilisateur +i) et qui n'ont pas de canal en commun avec le client émettant la requête) sont listés. Le même résultat peut être obtenu en utilisant le <nom> "0" ou tout joker correspondant à toutes les entrées possibles.

Le <nom> passé en paramètre est mis en correspondance avec les hôtes des utilisateurs, leurs véritables noms, et leurs pseudonymes si le canal <nom> n'est pas trouvé.

Si le paramètre "o" est passé, seuls les opérateurs sont listés, et ce, en fonction du masque fourni.

Réponses numériques :

```
ERR_NOSUCHSERVER
RPL_WHOREPLY                    RPL_ENDOFWHO
```

Exemples:

`WHO *.fi` ; Liste tous les utilisateurs qui correspondent à "*.fi".
`WHO jto* o` ; Liste tous les utilisateurs qui correspondent à "jto*", s'ils sont opérateurs.

4.5.2 Requête WHOIS

Commande: WHOIS
Paramètres: [<serveur>] <masque de pseudo>[,<masque de pseudo>[,...]]

Ce message est utilisé pour obtenir des informations sur un utilisateur donné. Le serveur répondra à ce message avec des messages numériques indiquant les différents statuts de chacun des utilisateurs qui correspondent au <masque de pseudo> (si vous pouvez les voir). S'il n'y a pas de joker dans le <masque de pseudo>, toutes les informations auxquelles vous avez accès au sujet de ce pseudo seront présentées. On peut séparer la liste des pseudonymes avec une virgule (',').

La dernière version envoie une requête à un serveur spécifique. C'est utile si vous voulez savoir depuis combien de temps l'utilisateur concerné a été oisif, car seul le serveur local (celui auquel cet utilisateur est directement connecté) connaît cette information, alors que tout le reste est connu par tous les serveurs.

Réponses numériques :

```
ERR_NOSUCHSERVER                ERR_NONICKNAMEGIVEN
RPL_WHOISUSER                   RPL_WHOISCHANNELS
RPL_WHOISCHANNELS               RPL_WHOISSERVER
RPL_AWAY                        RPL_WHOISOPERATOR
RPL_WHOISIDLE                   ERR_NOSUCHNICK
RPL_ENDOFWHOIS
```

Exemples:

`WHOIS wiz` ; renvoie les informations disponibles sur le pseudo WiZ
`WHOIS eff.org trillian` ; demande au serveur eff.org les informations concernant trillian

4.5.3 WHOWAS

Commande: WHOWAS
Paramètres: <pseudonyme> [<compte> [<serveur>]]

WHOWAS permet de demander des informations concernant un utilisateur qui n'existe plus. Cela peut être dû à un changement de pseudonyme ou au fait que l'utilisateur ait quitté l'IRC. En réponse à cette requête, le serveur cherche un pseudo correspondant dans l'historique des pseudonymes (sans utiliser de jokers). L'historique est parcouru à l'envers, de façon à renvoyer l'entrée la plus récente en premier. S'il y a plusieurs entrées, jusqu'à <compte> entrées seront retournées (ou toutes si le paramètre <compte> n'est pas donné). Si le nombre passé n'est pas positif, une recherche complète a lieu.

Réponses numériques :

```
ERR_NONICKNAMEGIVEN             ERR_WASNOSUCHNICK
RPL_WHOWASUSER                  RPL_WHOISSERVER
RPL_ENDOFWHOWAS
```

Exemples:

`WHOWAS Wiz` ; renvoie toutes les informations dans l'historique des pseudos au sujet du pseudo "WiZ";
`WHOWAS Mermaid 9` ; renvoie, au maximum, les neufs entrées les plus récentes dans l'historique des pseudos pour "Mermaid";
`WHOWAS Trillian 1 *.edu` ; demande l'entrée la plus récente pour "Trillian" au premier serveur trouvé qui correspond à "*.edu".

4.6 Messages divers

Les messages de cette catégorie ne font partie d'aucune des catégories ci-dessus, mais font néanmoins partie intégrante du protocole, et sont indispensables.

4.6.1 Message KILL

Commande: KILL
Paramètres: <pseudonyme> <commentaire>

Le message KILL est utilisé pour provoquer la fermeture de la connexion client/serveur par le serveur qui gère cette connexion. KILL est aussi utilisé par les serveurs qui rencontrent un doublon dans la liste des entrées de pseudonymes valides, afin de retirer les deux entrées. Elle est également accessible aux opérateurs.

Les clients qui ont des reconnexions automatiques rendent cette commande inefficace, car la déconnexion est brève. Cela permet tout de même d'interrompre un flux de données et est utile pour arrêter un flux abusif (trop important). Tout utilisateur peut demander à recevoir les messages KILL générés pour d'autres clients afin de garder un oeil sur les fauteurs de trouble éventuels.

Dans une arène où les pseudonymes doivent être globalement uniques, les messages KILL sont envoyés à chaque fois qu'un doublon est détecté (c'est-à-dire une tentative d'enregistrer deux utilisateurs avec le même pseudonyme) dans l'espoir qu'ils disparaîtront tous les deux, et qu'un seul réapparaîtra.

Le commentaire doit refléter la véritable raison du KILL. Pour les messages issus de serveurs, il est habituellement constitué des détails concernant les origines des deux pseudonymes en conflit. Les utilisateurs, eux, sont libres de fournir une raison adéquate, de façon à satisfaire ceux qui le voient. Afin de prévenir/d'éviter des KILL maquillés pour cacher l'identité de l'auteur d'être générés, le commentaire contient également un 'chemin de KILL' qui est mis à jour par tous les serveurs par lequel il passe, chacun ajoutant son nom au chemin.

Réponses numériques :

```
ERR_NOPRIVILEGES                ERR_NEEDMOREPARAMS
ERR_NOSUCHNICK                  ERR_CANTKILLSERVER
```

Exemple:

`KILL David (csd.bu.edu <- tolsun.oulu.fi)` ; Collision de pseudonymes entre csd.bu.edu et tolson.oulu.fi

NOTE:
Il est recommandé que seuls les opérateurs soient autorisés à déconnecter d'autres utilisateurs avec un message KILL. Dans un monde parfait, même les opérateurs ne devraient pas avoir besoin de cette commande, et on laisserait les serveurs se débrouiller.

4.6.2 Message PING

Commande: PING
Paramètres: <serveur1> [<serveur2>]

Le message PING est utilisé pour tester la présence d'un client actif à l'autre bout de la connexion. Un message PING est envoyé régulièrement si aucune activité n'est détectée sur une connexion. Si la connexion ne répond pas à la commande PING dans un certain délai, la connexion est fermée.

Tout client qui reçoit un message PING doit répondre au <serveur1> (serveur qui a envoyé le message PING) aussi rapidement que possible, avec un message PONG approprié pour indiquer qu'il est toujours là et actif. Les serveurs ne doivent pas répondre aux commandes PING, mais se fier au PING dans l'autre sens pour indiquer que la connexion est toujours active. Si le paramètre <serveur2> est spécifié, le message PING lui est transmis.

Réponses numériques :

```
ERR_NOORIGIN                    ERR_NOSUCHSERVER
```

Exemples:

`PING tolsun.oulu.fi` ; serveur envoyant un message PING à un autre serveur pour indiquer qu'il est toujours actif.
`PING WiZ` ; message PING envoyé au pseudo WiZ

4.6.3 Message PONG

Commande: PONG
Paramètres: <démon> [<démon2>]

Le message PONG est la réponse à un message PING. Si le paramètre <démon2> est donné, le message doit être transmis au démon donné. Le paramètre <démon> est le nom du démon responsable du message PING généré.

Réponses numériques :

```
ERR_NOORIGIN                    ERR_NOSUCHSERVER
```

Exemples:

`PONG csd.bu.edu tolsun.oulu.fi` ; message PONG de csd.bu.edu à tolsun.oulu.fi

4.6.4 Message ERROR

Commande: ERROR
Paramètres: < message d'erreur>

La commande ERROR est utilisée par les serveurs pour rapporter une erreur sérieuse ou fatale à ses opérateurs. Elle peut aussi être envoyée d'un serveur à un autre, mais ne doit pas être acceptée de simples clients inconnus.

Un message ERROR ne doit être utilisé que pour annoncer les erreurs qui ont lieu sur un lien serveur/serveur. Un message ERROR est envoyé au serveur associé (qui le transmet à tous ses opérateurs connectés) et à tous les opérateurs connectés. Il ne doit pas être transmis aux autres serveurs s'il est reçu d'un serveur.

Quand un serveur transmet un message ERROR à ses opérateurs, le message doit être encapsulé dans un message NOTICE, en indiquant que le client n'est pas responsable de l'erreur.

Réponses numériques :

```
Aucune.
```

Exemples:

`ERROR :Server *.fi already exists` ; message ERROR à l'autre serveur qui a provoqué cette erreur.
`NOTICE WiZ :ERROR from csd.bu.edu -- Server *.fi already exists` ; Même message ERROR qu'au dessus, mais envoyé à l'utilisateur Wiz sur l'autre serveur.

5. Messages optionnels

Cette section décrit les messages optionnels. Ils ne sont pas requis dans les implémentations des serveurs décrits ici. En l'absence de l'option, un message d'erreur doit être généré, ou une erreur commande inconnue. Si le message est destiné à un autre serveur, il doit être transmis (traitement de base obligatoire). Les nombres alloués pour cela sont listés avec les messages ci-dessous.

5.1 AWAY

Commande: AWAY
Paramètres: [message]

Avec le message AWAY, les clients peuvent définir une chaîne de réponse automatique pour toute commande PRIVMSG qui leur est destinée (et non pas à un canal sur lequel ils sont). La réponse est envoyée directement par le serveur au client envoyant une commande PRIVMSG. Le seul serveur à répondre est celui sur lequel le client émetteur est situé.

Le message AWAY est utilisé soit avec un paramètre (pour définir un message AWAY) ou sans (pour retirer le message AWAY).

Réponses numériques :

```
RPL_UNAWAY                      RPL_NOWAWAY
```

Exemples:

`AWAY :Parti déjeuner. De retour à 2 heures.` ; définit le message d'absence en "Parti déjeuner. De retour à 2 heures.".
`:WiZ AWAY` ; supprime l'absence de WiZ.

5.2 Message REHASH

Commande: REHASH
Paramètres: Aucun

Le massage REHASH est utilisé par les opérateurs pour forcer un serveur à relire et traiter son fichier de configuration.

Réponses numériques :

```
RPL_REHASHING                   ERR_NOPRIVILEGES
```

Exemples:

`REHASH` ; message d'un client ayant un statut d'opérateur au serveur, lui demandant de relire son fichier de configuration.

5.3 Message RESTART

Commande: RESTART
Paramètres: Aucun

Le message RESTART n'est utilisable que par un opérateur. Il sert à redémarrer le serveur. La gestion de ce message est optionnelle, car il est risqué de permettre à des personnes se connectant comme opérateur d'exécuter cette commande, qui cause une interruption de service (au moins).

La commande RESTART doit toujours être traitée par le serveur qui la reçoit, et non passée à un autre serveur.

Réponses numériques :

```
ERR_NOPRIVILEGES
```

Exemples:

`RESTART` ; pas de paramètres

5.4 Message SUMMON

Commande: SUMMON
Paramètres: <utilisateur> [<serveur>]

La commande SUMMON peut être utilisée pour envoyer à des utilisateurs qui sont sur l'hôte sur lequel s'exécute le serveur IRC un message leur demandant de joindre l'IRC. Ce message ne peut être envoyé que si le serveur (a) a la commande SUMMON activée, et (b) si le processus serveur peut écrire sur le tty (ou similaire) de l'utilisateur.

Si le paramètre <serveur> n'est pas donné, cela essaie d'appeler l'<utilisateur> du serveur sur lequel le client est connecté.

Si le SUMMON est désactivé sur un serveur, il doit renvoyer la réponse numérique ERR_SUMMONDISABLED et transmettre le message SUMMON.

Réponses numériques :

```
ERR_NORECIPIENT                 ERR_FILEERROR
ERR_NOLOGIN                     ERR_NOSUCHSERVER
RPL_SUMMONING
```

Exemples:

`SUMMON jto` ; appelle l'utilisateur jto sur l'hôte du serveur
`SUMMON jto tolsun.oulu.fi` ; appelle l'utilisateur jto sur l'hôte sur lequel le serveur "tolsun.oulu.fi" est lancé.

5.5 Commande USERS

Commande: USERS
Paramètres: [<serveur>]

La commande USERS fonctionne de façon similaire à WHO(1), RUSERS(1) et FINGER(1). Certains peuvent désactiver cette commande sur leur serveur pour des raisons de sécurité. En cas de désactivation, cela doit être indiqué par le retour de réponse numérique appropriée.

Réponses numériques :

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

Réponse de désactivation :

```
ERR_USERSDISABLED
```

Exemples:

`USERS eff.org` ; requiert la liste des utilisateurs connectés au serveur eff.org
`:John USERS tolsun.oulu.fi` ; requête de John pour obtenir la liste des utilisateurs du serveur tolsun.oulu.fi

5.6 Message WALLOPS

Commande: WALLOPS
Paramètres: Texte à envoyer à tous les opérateurs actuellement connectés.

Envoie un message à tous les opérateurs actuellement connectés. Après avoir essayé de laisser accès à cette commande à tous les utilisateurs, il a été constaté qu'on en abusait comme un moyen d'envoyer des messages à plein de personnes (comme WALL). A cause de cela, il est recommandé que l'implémentation courante de WALLOPS ne reconnaisse que les serveurs comme émetteurs de WALLOPS.

Réponses numériques :

```
ERR_NEEDMOREPARAMS
```

Exemples:

`:csd.bu.edu WALLOPS :Connect '*.uiuc.edu 6667' from Joshua` ; message WALLOPS de csd.bu.edu annonçant un message CONNECT reçu et traité, issu de Joshua.

5.7 Message USERHOST

Commande: USERHOST
Paramètres: <pseudonyme>{<espace ><pseudonyme>}

La commande USERHOST prend jusqu'à 5 pseudonymes, séparés par des virgules, et renvoie une liste d'informations pour chacun des pseudonymes qu'il a trouvés. La liste des réponses contient chaque réponse séparée par des espaces.

Réponses numériques :

```
RPL_USERHOST                    ERR_NEEDMOREPARAMS
```

Exemples:

`USERHOST Wiz Michael Marty p` ;requête USERHOST pour information sur les pseudos "Wiz", "Michael", "Marty" et "p"

5.8 Message ISON

Commande: ISON
Paramètres: <pseudonyme>{<espace><pseudonyme>}

La commande ISON a été implémentée pour fournir une manière rapide et efficace de savoir si un pseudonyme donné est connecté à l'IRC. ISON prend un (1) paramètre : une liste de pseudonymes séparés par des espaces. Chaque pseudonyme présent est ajouté à la chaîne de réponse du serveur. Ainsi, la chaîne de réponse peut être vide (aucun utilisateur n'est présent), une copie exacte de la chaîne de caractères passée en paramètres (ils sont tous présents), ou un sous-ensemble du groupe de pseudonymes passé en paramètre. La seule limite au nombre de pseudos qui peuvent être testés est la troncature des commandes à une longueur de 512 caractères.

ISON n'est traitée que par le serveur local au client effectuant la requête, et n'est donc pas passée pour traitement aux autres serveurs

Réponses numériques :

```
RPL_ISON                ERR_NEEDMOREPARAMS
```

Exemples:

`ISON phone trillian WiZ jarlek Avalon Angel Monstah` ; Exemple de requête ISON pour 7 pseudonymes

6. Réponses

Ce qui suit est une liste de réponses numériques générées à la suite des commandes spécifiées ci-dessus. Chaque réponse numérique est donnée avec son numéro, son nom, et sa chaîne de réponse (en anglais).

6.1 Réponses d'erreur

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

Utilisé pour indiquer que le pseudonyme passé en paramètre à la commande n'est pas actuellement utilisé.

402 ERR_NOSUCHSERVER

"<nom de serveur> :No such server"

Utilisé pour indiquer que le nom du serveur donné n'existe pas actuellement.

403 ERR_NOSUCHCHANNEL
"<nom de canal> :No such channel"

Utilisé pour indiquer que le nom de canal donné est invalide.

404 ERR_CANNOTSENDTOCHAN
"<nom de canal> :Cannot send to channel"

Envoyé à un utilisateur qui (a) soit n'est pas dans un canal en mode +n ou (b) n'est pas opérateur (ou mode +v) sur un canal en mode +m ; et essaie d'envoyer un PRIVMSG à ce canal.

405 ERR_TOOMANYCHANNELS
"<nom de canal> :You have joined too many channels"

Envoyé à un utilisateur quand il a atteint le nombre maximal de canaux qu'il est autorisé à accéder simultanément, s'il essaie d'en rejoindre un autre.

406 ERR_WASNOSUCHNICK
"<nom de canal> :There was no such nickname"

Renvoyé par WHOWAS pour indiquer qu'il n'y a pas d'information dans l'historique concernant ce pseudonyme.

407 ERR_TOOMANYTARGETS
"<destination> :Duplicate recipients. No message delivered"

Renvoyé à un client qui essaie d'envoyer un PRIVMSG/NOTICE utilisant le format de destination utilisateur@hôte pour lequel utilisateur@hôte a plusieurs occurrences.

409 ERR_NOORIGIN
":No origin specified"

Message PING ou PONG sans le paramètre origine qui est obligatoire puisque ces commandes doivent marcher sans préfixe.

411 ERR_NORECIPIENT
":No recipient given (<commande>)"

Pas de destinataire.

412 ERR_NOTEXTTOSEND
":No text to send"

Pas de texte à envoyer.

413 ERR_NOTOPLEVEL
"<masque> :No toplevel domain specified"

Domaine principal non spécifié.

414 ERR_WILDTOPLEVEL
"<masque> :Wildcard in toplevel domain"

Joker dans le domaine principal

Les erreurs 412-414 sont renvoyées par PRIVMSG pour indiquer que le message n'a pas été délivré. ERR_NOTOPLEVEL et ERR_WILDTOPLEVEL sont les erreurs renvoyées lors d'une utilisation invalide de "PRIVMSG $<serveur>" ou de "PRIVMSG #<hôte>".

421 ERR_UNKNOWNCOMMAND
"<commande> :Unknown command"

Renvoyé à un client enregistré pour indiquer que la commande envoyée est inconnue du serveur.

422 ERR_NOMOTD
":MOTD File is missing"

Le fichier MOTD du serveur n'a pas pu être ouvert.

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

Renvoyé par un serveur en réponse à un message ADMIN quand il y a une erreur lors de la recherche des informations appropriées.

424 ERR_FILEERROR
":File error doing <opération> on <fichier>"

Message d'erreur générique utilisé pour rapporter un échec d'opération de fichier durant le traitement d'un message.

431 ERR_NONICKNAMEGIVEN
":No nickname given"

Renvoyé quand un paramètre pseudonyme attendu pour une commande n'est pas fourni.

432 ERR_ERRONEUSNICKNAME
"<pseudo> :Erroneus nickname"

Renvoyé après la réception d'un message NICK qui contient des caractères qui ne font pas partie du jeu autorisé. Voir les sections 1 et 2.2 pour les détails des pseudonymes valides.

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

Renvoyé quand le traitement d'un message NICK résulte en une tentative de changer de pseudonyme en un déjà existant.

436 ERR_NICKCOLLISION
"<nick> :Nickname collision KILL"

Renvoyé par un serveur à un client lorsqu'il détecte une collision de pseudonymes (enregistrement d'un pseudonyme qui existe déjà sur un autre serveur).

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

Renvoyé par un serveur pour indiquer que l'utilisateur donné n'est pas dans le canal spécifié.

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

Renvoyé par le serveur quand un client essaie une commande affectant un canal dont il ne fait pas partie.

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

Renvoyé quand un client essaie d'inviter un utilisateur sur un canal où il est déjà.

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

Renvoyé par un SUMMON si la commande n'a pas pu être accomplie, car l'utilisateur n'est pas connecté.

445 ERR_SUMMONDISABLED
":SUMMON has been disabled"

Renvoyé en réponse à une commande SUMMON si le SUMMON est désactivé. Tout serveur qui ne gère pas les SUMMON doit retourner cette valeur.

446 ERR_USERSDISABLED
":USERS has been disabled"

Retourné en réponse à une commande USERS si la commande est désactivée. Tout serveur qui ne gère pas les USERS doit retourner cette valeur.

451 ERR_NOTREGISTERED
":You have not registered"

Retourné par le serveur pour indiquer à un client qu'il doit être enregistré avant que ses commandes soient traitées.

461 ERR_NEEDMOREPARAMS
"<commande> :Not enough parameters"

Renvoyé par un serveur par de nombreuses commandes, afin d'indiquer que le client n'a pas fourni assez de paramètres.

462 ERR_ALREADYREGISTRED
":You may not reregister"

Retourné par le serveur à tout lien qui tente de changer les détails enregistrés (tels que mot de passe et détails utilisateur du second message USER)

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

Renvoyé à un client qui essaie de s'enregistrer sur un serveur qui n'accepte pas les connexions depuis cet hôte.

464 ERR_PASSWDMISMATCH
":Password incorrect"

Retourné pour indiquer l'échec d'une tentative d'enregistrement d'une connexion dû à un mot de passe incorrect ou manquant.

465 ERR_YOUREBANNEDCREEP
":You are banned from this server"

Retourné après une tentative de connexion et d'enregistrement sur un serveur configuré explicitement pour vous refuser les connexions.

467 ERR_KEYSET
"<canal> :Channel key already set"

Clé de canal déjà définie.

471 ERR_CHANNELISFULL
"<canal> :Cannot join channel (+l)"

Impossible de joindre le canal (+l)

472 ERR_UNKNOWNMODE
"<caractère> :is unknown mode char to me"

Mode inconnu.

473 ERR_INVITEONLYCHAN
"<canal> :Cannot join channel (+i)"

Impossible de joindre le canal (+i).

474 ERR_BANNEDFROMCHAN
"<canal> :Cannot join channel (+b)"

Impossible de joindre le canal (+b).

475 ERR_BADCHANNELKEY
"<canal> :Cannot join channel (+k)"

Impossible de joindre le canal (+k).

481 ERR_NOPRIVILEGES
":Permission Denied- You're not an IRC operator"

Toute commande qui requiert le privilège d'opérateur pour opérer doit retourner cette erreur pour indiquer son échec.

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

Toute commande qui requiert les privilèges 'chanop' (tels les messages MODE) doit retourner ce message à un client qui l'utilise sans être chanop sur le canal spécifié.

483 ERR_CANTKILLSERVER
":You cant kill a server!"

Toute tentative d'utiliser la commande KILL sur un serveur doit être refusée et cette erreur renvoyée directement au client.

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

Si un client envoie un message OPER et que le serveur n'a pas été configuré pour autoriser les connexions d'opérateurs de cet hôte, cette erreur doit être retournée.

501 ERR_UMODEUNKNOWNFLAG
":Unknown MODE flag"

Renvoyé par un serveur pour indiquer que le message MODE a été envoyé avec un pseudonyme et que le mode spécifié n'a pas été identifié.

502 ERR_USERSDONTMATCH
":Cant change mode for other users"

Erreur envoyée à tout utilisateur qui essaie de voir ou de modifier le mode utilisateur d'un autre client.

6.2 Réponses aux commandes.

300 RPL_NONE

Numéro de réponse bidon. Inutilisé.

302 RPL_USERHOST
":[<réponse>{<espace><réponse>}]"

Format de réponse utilisé par USERHOST pour lister les réponses à la liste des requêtes. La chaîne de réponse est composée ainsi :
<réponse> ::= <pseudo>['*'] '=' <'+'|'-'><hôte>
Le '*' indique si le client est enregistré comme opérateur. Les caractères '-' ou '+' indiquent respectivement si le client a défini un message AWAY ou non.

303 RPL_ISON
":[<pseudo> {<espace><pseudo>}]"

Format de réponse utilisé par ISON pour lister les réponses à la liste de requête.

301 RPL_AWAY
"<pseudo> :<message d'absence>"
305 RPL_UNAWAY
":You are no longer marked as being away"
306 RPL_NOWAWAY
":You have been marked as being away"

Ces trois réponses sont utilisées avec la commande AWAY (si elle est autorisée). RPL_AWAY est envoyé à tout client qui envoie un PRIVMSG à un client absent. RPL_AWAY n'est envoyé que par le serveur sur lequel le client est connecté. Les réponses RPL_UNAWAY et RPL_NOWAWAY sont envoyées quand un client retire et définit un message AWAY.

311 RPL_WHOISUSER
"<pseudo> <utilisateur> <hôte> * :<vrai nom>"
312 RPL_WHOISSERVER
"<pseudo> <serveur> :<info serveur>"
313 RPL_WHOISOPERATOR
"<pseudo> :is an IRC operator"
317 RPL_WHOISIDLE
"<pseudo> <integer> :seconds idle"
318 RPL_ENDOFWHOIS
"<pseudo> :End of /WHOIS list"
319 RPL_WHOISCHANNELS
"<pseudo> :{[@|+]<canal><espace>}"

Les réponses 311 à 313 et 317 à 319 sont toutes générées en réponse à un message WHOIS. S'il y a assez de paramètres, le serveur répondant doit soit formuler une réponse parmi les numéros ci-dessus (si le pseudo recherché a été trouvé) ou renvoyer un message d'erreur. Le '*' dans RPL_WHOISUSER est là en tant que littéral et non en tant que joker. Pour chaque jeu de réponses, seul RPL_WHOISCHANNELS peut apparaître plusieurs fois (pour les longues listes de noms de canaux). Les caractères '@' et '+' à côté du nom de canal indiquent si un client est opérateur de canal, ou si on l'a autorisé à parler dans un canal modéré. La réponse RPL_ENDOFWHOIS est utilisée pour marquer la fin de la réponse WHOIS.

314 RPL_WHOWASUSER
"<pseudo> <utilisateur> <hôte> * :<vrai nom>"
369 RPL_ENDOFWHOWAS
"<pseudo> :End of WHOWAS"

Lorsqu'il répond à un message WHOWAS, un serveur doit utiliser RPL_WHOWASUSER, RPL_WHOISSERVER ou ERR_WASNOSUCHNICK pour chacun des pseudonymes de la liste fournie. A la fin de toutes les réponses, il doit y avoir un RPL_ENDOFWHOWAS (même s'il n'y a eu qu'une réponse, et que c'était une erreur).

321 RPL_LISTSTART
"Channel :Users Name"
322 RPL_LIST
"<canal> <# visible> :<sujet>"
323 RPL_LISTEND
":End of /LIST"

Les réponses RPL_LISTSTART, RPL_LIST, RPL_LISTEND marquent le début, les réponses proprement dites, et la fin du traitement d'une commande LIST. S'il n'y a aucun canal disponible, seules les réponses de début et de fin sont envoyées.

324 RPL_CHANNELMODEIS
"<canal> <mode> <paramètres de mode>"
331 RPL_NOTOPIC
"<canal> :No topic is set"
332 RPL_TOPIC
"<canal> :<sujet>"

Lors de l'envoi d'un message TOPIC pour déterminer le sujet d'un canal, une de ces deux réponses est envoyée. Si le sujet est défini, RPL_TOPIC est renvoyée, sinon c'est RPL_NOTOPIC.

341 RPL_INVITING
"<canal> <pseudo>"

Renvoyé par un serveur pour indiquer que le message INVITE a été enregistré, et est en cours de transmission au client final.

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

Renvoyé par un serveur en réponse à un message SUMMON pour indiquer qu'il appelle cet utilisateur.

351 RPL_VERSION
"<version>.<debuglevel> <serveur> :<commentaires>"

Réponse du serveur indiquant les détails de sa version. <version> est la version actuelle du programme utilisé (comprenant le numéro de mise à jour) et <debuglevel> est utilisé pour indiquer si le serveur fonctionne en mode débugage.
Le champ <commentaire> peut contenir n'importe quel commentaire au sujet de la version, ou des détails supplémentaires sur la version.

352 RPL_WHOREPLY
"<canal> <utilisateur> <hôte> <serveur> <pseudo> <H|G>[*][@|+] :<compteur de distance> <vrai nom>"
315 RPL_ENDOFWHO
"<nom> :End of /WHO list"

La paire RPL_WHOREPLY et RPL_ENDOFWHO est utilisée en réponse à un message WHO. Le RPL_WHOREPLY n'est envoyé que s'il y a une correspondance à la requête WHO. S'il y a une liste de paramètres fournie avec le message WHO, un RPL_ENDOFWHO doit être envoyé après le traitement de chaque élément de la liste, <nom> étant l'élément.

353 RPL_NAMREPLY
"<canal> :[[@|+]<pseudo> [[@|+]<pseudo> [...]]]"
366 RPL_ENDOFNAMES
"<canal> :End of /NAMES list"

En réponse à un message NAMES, une paire consistant de RPL_NAMREPLY et RPL_ENDOFNAMES est renvoyée par le serveur au client. S'il n'y a pas de canal résultant de la requête, seul RPL_ENDOFNAMES est retourné. L'exception à cela est lorsqu'un message NAMES est envoyé sans paramètre et que tous les canaux et contenus visibles sont renvoyés en une suite de message RPL_NAMEREPLY avec un RPL_ENDOFNAMES indiquant la fin.

364 RPL_LINKS
"<masque> <serveur> :<compteur de distance> <info serveur>"
365 RPL_ENDOFLINKS
"<mask> :End of /LINKS list"

En réponse à un message LINKS, un serveur doit répondre en utilisant le nombre RPL_LINKS, et indiquer la fin de la liste avec une réponse RPL_ENDOFLINKS.

367 RPL_BANLIST
"<canal> <identification de bannissement>"
368 RPL_ENDOFBANLIST
"<canal> :End of channel ban list"

Quand il liste les bannissements actifs pour un canal donné, un serveur doit renvoyer la liste en utilisant les messages RPL_BANLIST et RPL_ENDOFBANLIST. Un RPL_BANLIST différent doit être utilisé pour chaque identification de bannissement. Après avoir listé les identifications de bannissement (s'il y en a), un RPL_ENDOFBANLIST doit être renvoyé.

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

Un serveur répondant à un message INFO doit envoyer toute sa série d'info en une suite de réponses RPL_INFO, avec un RPL_ENDOFINFO pour indiquer la fin des réponses.

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

Lorsqu'il répond à un message MOTD et que le fichier MOTD est trouvé, le fichier est affiché ligne par ligne, chaque ligne ne devant pas dépasser 80 caractères, en utilisant des réponses au format RPL_MOTD. Celles-ci doivent être encadrées par un RPL_MOTDSTART (avant les RPL_MOTDs) et un RPL_ENDOFMOTD (après).

381 RPL_YOUREOPER
":You are now an IRC operator"

RPL_YOUREOPER est renvoyé à un client qui vient d'émettre un message OPER et a obtenu le statut d'opérateur.

382 RPL_REHASHING
"<fichier de configuration> :Rehashing"

Si l'option REHASH est activée et qu'un opérateur envoie un message REHASH, un RPL_REHASHING est renvoyé à l'opérateur.

391 RPL_TIME
"<serveur> :<chaîne indiquant l'heure locale du serveur>"

Lorsqu'il répond à un message TIME, un serveur doit répondre en utilisant le format RPL_TIME ci-dessus. La chaîne montrant l'heure ne doit contenir que le jour et l'heure corrects. Il n'y a pas d'obligation supplémentaire.

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

Si le message USERS est géré par un serveur, les réponses RPL_USERSTART, RPL_USERS, RPL_ENDOFUSERS et RPL_NOUSERS sont utilisées. RPL_USERSSTART doit être envoyé en premier, suivi par soit une séquence de RPL_USERS soit un unique RPL_NOUSER. Enfin, vient un RPL_ENDOFUSERS.

200 RPL_TRACELINK
"Link <version & niveau de débugage> <destination> <serveur suivant>"
201 RPL_TRACECONNECTING
"Try. <classe> <serveur>"
202 RPL_TRACEHANDSHAKE
"H.S. <classe> <serveur>"
203 RPL_TRACEUNKNOWN
"???? <classe> [<adresse IP du client au format utilisant des points>]"
204 RPL_TRACEOPERATOR
"Oper <classe> <pseudo>"
205 RPL_TRACEUSER
"User <classe> <pseudo>"
206 RPL_TRACESERVER
"Serv <classe> <int>S <int>C <serveur> <pseudo!utilisateur|*!*>@<hôte|serveur>"
208 RPL_TRACENEWTYPE
"<nouveau type> 0 <nom du client>"
261 RPL_TRACELOG
"File <fichier log> <niveau de débugage>"

Les RPL_TRACE* sont tous renvoyés par le serveur en réponse à un message TRACE. Le nombre de messages retournés dépend de la nature du message TRACE, et s'il a été envoyé par un opérateur ou non. Il n'y a pas d'ordre définissant lequel doit être le premier. Les réponses RPL_TRACEUNKNOWN, RPL_TRACECONNECTING et RPL_TRACEHANDSHAKE sont toutes utilisées pour des connexions qui n'ont pas été complètement établies, et sont soit inconnues, soit toujours en cours de connexion, soit dans la phase terminale de la 'poignée de main du serveur'. RPL_TRACELINK est envoyé par tout serveur qui traite un message TRACE et doit le transmettre à un autre serveur. La liste de RPL_TRACELINK envoyée en réponse à une commande TRACE traversant le réseau IRC devrait refléter la connectivité actuelle des serveurs le long du chemin. RPL_TRACENEWTYPE est utilisé pour toute connexion qui n'entre pas dans les autres catégories, mais qui est néanmoins affichée.

211 RPL_STATSLINKINFO
"<nom du lien> <sendq> <messages envoyés> <octets envoyés> <message reçus> <octets reçus> <temps de connexion>"
212 RPL_STATSCOMMANDS
"<commande> <compteur>"
213 RPL_STATSCLINE

"C <hôte> * <nom> <port> <classe>"
214 RPL_STATSNLINE
"N <hôte> * <nom> <port> <classe>"
215 RPL_STATSILINE
"I <hôte> * <hôte> <port> <classe>"
216 RPL_STATSKLINE
"K <hôte> * <nom d'utilisateur> <port> <classe>"
218 RPL_STATSYLINE
"Y <classe> <fréquence des PINGS> <fréquence de connexion> <sendq max>"
219 RPL_ENDOFSTATS
"<lettre de stats> :End of /STATS report"
241 RPL_STATSLLINE
"L <masque d'hôte> * <nom de serveur> <profondeur maxi>"
242 RPL_STATSUPTIME
":Server Up %d days %d:%02d:%02d"
243 RPL_STATSOLINE
"O <masque d'hôte> * <nom>"
244 RPL_STATSHLINE
"H <masque d'hôte> * <nom de serveur>"

221 RPL_UMODEIS
"<chaîne de mode utilisateur>"

Pour répondre à une requête au sujet du mode du client, RPL_UMODEIS est renvoyé.

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

Lors du traitement d'un message LUSERS, le serveur envoie un lot de réponses parmi RPL_LUSERCLIENT, RPL_LUSEROP, RPL_USERUNKNOWN, RPL_LUSERCHANNELS et RPL_LUSERME. Lorsqu'il répond, un serveur doit envoyer RPL_LUSERCLIENT et RPL_LUSERME. Les autres réponses ne sont renvoyées que si le nombre trouvé n'est pas nul.

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

Lorsqu'il répond à un message ADMIN, un serveur doit renvoyer les réponses RLP_ADMINME à RPL_ADMINEMAIL et fournir un texte de message avec chacune. Pour RPL_ADMINLOC1, on attend une description de la ville et de l'état où se trouve le serveur, suivie des détails de l'université et du département (RPL_ADMINLOC2), et finalement le contact administratif pour ce serveur (avec obligatoirement une adresse email) dans RPL_ADMINEMAIL.

6.3 Nombres réservés.

Ces nombres ne sont pas décrits ci-dessus parce qu'ils tombent dans l'une des catégories suivantes :

1. Plus utilisés ;
2. Réservés à une future utilisation ;
3. Utilisés à l'heure actuelle, mais faisant partie des caractéristiques non génériques des serveurs IRC courants.

```
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. Authentification des clients et des serveurs

Les clients et serveurs sont tous deux soumis au même niveau d'authentification. Dans les deux cas, une recherche du nom d'hôte depuis l'adresse IP (avec vérification inverse) est effectuée pour toutes les connexions au serveur. Les deux connexions sont alors sujettes à une vérification de mot de passe (s'il y a un mot de passe défini pour cette connexion). Ces vérifications sont possibles pour toutes les connexions, bien que la vérification d'un mot de passe ne soit généralement utilisée que pour les serveurs.

Une vérification additionnelle de plus en plus commune est le nom d'utilisateur à l'origine de la connexion. Trouver le nom d'utilisateur à l'origine d'une connexion implique typiquement l'utilisation d'un serveur d'authentification tel que IDENT, comme il est décrit dans la RFC 1413.

Étant donné qu'il n'est pas facile d'établir avec fiabilité qui est à l'autre bout d'une connexion réseau, l'utilisation de mots de passe est fortement recommandée sur une connexion inter-serveur, en plus des autres mesures telles que l'utilisation d'un serveur d'identité.

8. Implémentations actuelles

La seule implémentation actuelle de ce protocole est le serveur IRC version 2.8. Les versions précédentes peuvent implémenter certaines ou toutes les commandes décrites dans ce document en utilisant les messages NOTICE à la place des réponses numériques. Malheureusement, à cause des problèmes de compatibilité ascendante, les implémentations de certaines parties de ce document diffèrent de ce qui est spécifié. Une différence notable est :
* La présence de tout LF ou tout CR dans le message marque sa fin (au lieu de la séquence préconisée CR-LF) ;

Le reste de cette section traite de questions qui sont pour la plupart intéressantes pour ceux qui veulent implémenter un serveur, mais certaines s'appliquent aussi directement aux clients.

8.1 Protocole Réseau: TCP - Pourquoi il est le plus approprié.

IRC a été implémenté sur TCP car TCP fournit un protocole réseau fiable qui est approprié à cette échelle de discussions. L'utilisation d'IP multicast est une alternative, mais n'est pas très répandue à l'heure actuelle.

8.1.1 Support des sockets Unix

Etant donné que les domaines de sockets Unix permettent les opérations listen/connect, les implémentations actuelles peuvent être configurées pour écouter et accepter aussi bien les clients que les serveurs sur un domaine de socket Unix. On reconnaît les sockets à un nom d'hôte commençant par un '/'.

Lors de la communication des informations au sujet d'un domaine de socket Unix, le serveur doit remplacer le nom de chemin par le vrai nom d'hôte, à moins que le vrai nom de socket soit demandé.

8.2 Traitement des commandes

Afin de fournir des E/S réseaux 'non-tamponnées' utiles aux clients et aux serveurs, à chaque connexion est associé son propre 'tampon d'entrée' dans lequel les résultats de lectures et traitements les plus récents sont conservés. Une taille de tampon de 512 octets est utilisée afin de contenir un message complet, bien qu'il en contienne habituellement plusieurs. Le tampon privé est traité après toute opération de lecture à la recherche de messages valides. Lors du traitement de messages multiples en provenance d'un client, on doit faire attention au cas où un des messages causerait le départ du client.

8.3 Distribution de message

Il est courant de trouver des liens réseaux saturés ou des hôtes à qui vous envoyez des données et qui sont incapables d'en faire autant. Bien qu'Unix gère typiquement cela à travers la fenêtre TCP et ses tampons internes, le serveur a généralement de grandes quantités de données à envoyer (spécialement lorsqu'une nouvelle connexion serveur/serveur est créée) et les petits tampons fournis dans le noyau ne sont pas suffisants à la queue de sortie. Pour alléger ce problème, une "queue d'envoi" est utilisée comme une queue FIFO pour les données à envoyer. Une "queue d'envoi" typique peut croître jusqu'à 200ko sur un gros réseau IRC, avec des connexions réseau lentes quand un nouveau serveur se connecte.

Lorsqu'il traite ses connexions, un serveur doit d'abord lire et traiter toutes les données entrantes, en mémorisant les données qui seront émises. Quand toutes les entrées disponibles sont traitées, la queue d'envoi est expédiée. Cela réduit le nombre d'appels systèmes write() et aide TCP à faire des paquets plus gros.

8.4 La vie d'une connexion

Pour détecter quand une connexion est morte ou ne répond plus, le serveur doit envoyer un PING à toutes les connexions dont il n'a pas eu de réponse depuis un temps donné.

Si une connexion ne répond pas à temps, elle est fermée en utilisant les procédures appropriées. Une connexion est également lâchée si son sendq grossit au-delà du maximum autorisé, car il vaut mieux fermer une connexion lente que d'avoir le processus serveur bloqué.

8.5 Établissement d'une connexion serveur à client

Lors de la connexion à un serveur IRC, on envoie au client le MOTD (s'il est présent) ainsi que le nombre actuel d'utilisateurs et de serveurs (comme pour la commande LUSER). Le serveur doit également envoyer un message non équivoque au client, qui stipule son nom, sa version, ainsi que tout autre message d'introduction qui lui semble approprié.

Après cela, le serveur doit envoyer le pseudo du nouvel utilisateur, et d'autres informations aussi bien fournies par lui-même (commande USER) que découvertes par le serveur (de la part des serveurs DNS et IDENT). Le serveur doit envoyer ces informations à la première commande NICK suivie de USER.

8.6 Établissement d'une connexion serveur/serveur

Le processus d'établissement d'une connexion serveur à serveur est plein de dangers, car il y a de nombreux domaines où un problème peut survenir *{ - the least of which are race conditions.}*

Après avoir reçu une connexion suivie d'une paire PASS/SERVER qui a été reconnue valide, le serveur doit répondre avec ses propres informations PASS/SERVER pour cette connexion, ainsi que toutes les informations d'état qu'il connaît comme décrit ci-dessous.

Quand le serveur initiant reçoit la paire PASS/SERVER, lui aussi vérifie que le serveur répondant est authentifié correctement avant d'accepter la connexion comme étant ce serveur.

8.6.1 Échange des informations d'état des serveurs à la connexion

L'ordre des informations d'état échangées entre les serveurs est essentiel. L'ordre requis est le suivant :

- tous les autres serveurs connus ;
- toutes les informations utilisateurs connues ;
- toutes les informations de canaux connues.

Les informations concernant les serveurs sont envoyées avec des messages SERVER supplémentaires, les informations utilisateurs avec des messages NICK/USER/MODE/JOIN et celles des canaux avec des messages MODE.

NOTE : Les sujets des canaux ne sont PAS échangés ici, car la commande TOPIC écrase toute information de sujet précédente, si bien que, au mieux, les deux côtés de la connexion échangeraient les sujets.

En passant les informations d'état concernant les serveurs en premier, toutes les collisions avec des serveurs qui existeraient déjà ont lieu avant les collisions de pseudo dues à un second serveur introduisant un pseudonyme particulier. En raison de l'obligation de réseau IRC à n'exister que sur un graphe acyclique, il est possible que le réseau se soit déjà reconnecté ailleurs, et l'endroit où la collision a lieu indique où le réseau doit être divisé.

8.7 Terminaison des connexions serveur/client.

Lorsqu'une connexion client se ferme, un message QUIT est généré de la part du client par le serveur sur lequel le client était connecté. Aucun autre message ne doit être généré ou utilisé.

8.8 Terminaison des connexions serveur/serveur.

Si une connexion serveur/serveur est fermée, soit par un SQUIT généré à distance, soit par une cause 'naturelle', le reste du réseau IRC doit le prendre en compte, et c'est au serveur qui détecte la fermeture de faire circuler l'information. Le serveur envoie une liste de SQUIT (un par serveur au-delà de la connexion coupée) et une liste de QUIT (un par client au-delà de la connexion coupée).

8.9 Suivi des changements de pseudonymes

Tous les serveurs IRC doivent garder un historique des changements récents de pseudonymes. Cela est nécessaire pour offrir au serveur la possibilité de garder le contact quand une commande concerne un utilisateur changeant de pseudo. Les commandes qui doivent vérifier un changement de pseudo sont :

- KILL (le pseudo se faisant tuer)
- MODE (+/- o,v)
- KICK (le pseudo se faisant exclure)

Aucune autre commande ne doit vérifier un changement de pseudo.

Dans les cas ci-dessus, le serveur doit tout d'abord vérifier l'existence du pseudonyme, puis vérifier l'historique pour voir à qui appartient ce pseudo. Cela réduit les chances de problèmes, mais ne les empêche pas complètement, ce qui peut résulter au final de l'affectation du mauvais client. Lors du traçage des changements de pseudonymes pour une des commandes ci-dessus, il est recommandé qu'un intervalle de temps soit donné, et que les entrées trop vieilles soient ignorées.

Pour un historique parfait, un serveur devrait être capable de garder les pseudonymes de tous les clients qui ont décidé d'un changement. La taille est limitée par d'autres facteurs (tels que la mémoire, ...)

8.10 Contrôle d'inondation des clients

Dans un gros réseau de serveurs IRC interconnectés, il est assez facile, pour un simple client connecté, d'émettre un flux continu de messages qui résultent non seulement en l'inondation du réseau, mais aussi en la dégradation de la qualité de service fournie aux autres clients. Au lieu de demander à chaque 'victime' de gérer sa propre protection, la protection contre les inondations est incluse dans le serveur et est appliquée à tous les clients, à l'exception des services. L'algorithme actuel est le suivant :

- vérifier si 'l'horodatage du message' du client est inférieur à l'heure actuelle (et le mettre à l'heure actuelle le cas échéant) ;