SIMON Hugo, MARIMANJAKA Darafify
Université Gustave Eiffel

Protocole ChatOS

Status du protocole

Ce RFC décrit un nouveau standard de communication entre un serveur et de multiple, et est sujet à des remarques pour l'amélioration de celui-ci.

Sommaire

Le protocole ChatOS est un protocole de communication entre deux clients ou plusieurs clients, la particularité par rapport à un serveur de chat standard comme IRC, est que deux clients peuvent demander à établir une connexion TCP entre eux sans qu'aucun des deux clients ne dévoile à l'autre son adresse IP. Celle ci sera uniquement connue par le serveur et les clients n'utilisent que les pseudos pour identifier le destinataire ou le client qui a envoyé le message.

Ce document décrit le protocole et le format des paquets qui seront transmis ainsi que les choix qui auront été fait pour ce format.

1. Objectif

Le but de ce protocole est de fournir un standard pour un système de chat en ligne permettant aux utilisateurs de discuter entre eux sans divulguer leur adresse ip, il permet la discussion privée entre deux utilisateurs au même titre que la discussion sur un chat “général” visible par tous.
Un utilisateur pourra donc :
Se connecter en fournissant un pseudo,
Une fois connecté, demander une connection privée avec un autre utilisateur,
envoyer un message en TCP dans un chat général
recevoir des messages en TCP,
se déconnecter,
Si la connexion privée est acceptée, il pourra envoyer des messages à un utilisateur de manière privée et en recevoir via un système de sockets.
Les données envoyées selon le type d'action souhaité et les réponses envoyées par le serveur sont détaillées plus bas

2. Aperçu du protocole

Pour que chacun des deux clients communiquent sans transmettre leurs adresses IP, chacun des deux clients se connectera avec un nouveau socket au serveur.
Par exemple pour deux clients A et B :
Le client A se connectera avec une socket SA et le client B, une socket SB. Après une phase d'authentification, le serveur relaiera tous les octets lus sur SA vers SB et tous les octets lus sur SB vers SA. Après cette phase d'authentification, les deux clients utilisent cette connexion de manière complètement transparente. Tout doit se passer comme s'il s'agissait d'une connexion TCP normale entre A et B.
Pour une discussion dans le chat général, le client envoie un message par TCP au serveur qui va le renvoyer à tous les utilisateurs connectés sur le chat général.

3. Relation aux autres protocole

Le protocole ChatOS est très dépendant des protocoles IP et TCP, les messages transitent tous via TCP et, dans le cadre de la fonction chat général, les adresses IP des clients sont utilisées lors des communications par le serveur, mais ne sont pas divulguées aux autres clients.
Il s’inspire aussi du protocole IRC pour le système de communication entre différents clients.

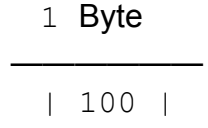
4. Protocole d’authentification

La première étape de notre protocole RFC est la connexion au serveur par un client. Pour ce faire, celui-ci va envoyer une requête de connexion commençant par l’opcode correspondant, ici 1, suivi d’un entier correspondant à la taille en octet du pseudo qui va être envoyé, suivi de son pseudo.
Si le pseudo n’est pas déjà utilisé, le serveur accepte la connexion et répond avec l’opcode 100.
Désormais le client est connecté au serveur et va donc pouvoir envoyer des messages aux autres clients connectés et recevoir leur réponse.

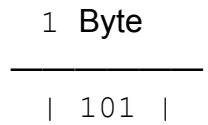
Voici la représentation d’une requête de connexion:

Byte	Int	String
1	pseudo Size	pseudo

Si l'authentification est réussie, le serveur va envoyer au client un octet correspondant à l'opcode 100 :



Si elle est refusée, le serveur va envoyer un octet correspondant à l'opcode 101 :



Ces octets seront ensuite lus par le client et il affichera un message en conséquence.

5. Paquets émis par ChatOS

Le protocole ChatOS permet d'émettre ou de recevoir les trames suivantes :

opcode	opération
0	Connexion au serveur
1	Déconnexion d'un client
2	Demande de connexion privée avec un autre client
4	Envoi d'un message (à tous les clients connectés)
5	Réception d'un message
6	Refus de connexion privée
7	Acceptation de connexion privée
8	Envoi message privé
9	Réception d'un message privé
100	Authentification réussi
101	Authentification refusé
104	Erreur rencontré

6. Délimitation des requêtes

pour qu'un client puisse envoyer plusieurs requêtes à la suite, nous avons fait le choix d'envoyer des entiers correspondant à la taille des données à lire avant chacune de ces données, ce qui permet de faire plusieurs échanges à la suite sans avoir à ouvrir de nouvelles connections et permet de plus facilement contrôler le fait que les données soient complètent qu'avec des marqueurs de fin par exemple.
De plus cela simplifie leur lecture étant donné que

7. Connexion privée entre deux clients

Lorsqu'un client fait la demande pour avoir une connexion privée il va envoyer une requête avec l' OPCODE 2 suivi d'un entier correspondant à la taille du pseudo de celui avec lequel il veut se connecter qui se finit par le pseudo en question.
Le serveur va ensuite transmettre cette demande a l'autre client.
Celui-ci va pouvoir refuser ou accepter.
La réponse sera transmise ensuite au demandeur et si celle-ci est acceptée le serveur va établir la connexion entre les deux clients.

Ci-dessous le format des paquets:

Byte	Int	String
2	targetPseudoSize	targetPseudo

8. Chat / Chat privé

lorsque le client envoie un message public, une trame au format suivant est envoyé contenant uniquement le message en question:

Byte	Int	String
4	messageSize	message

Le serveur transmet le message aux autres clients:

Byte	Int	String	Int	String
5	senderPseudoSize	senderPseudo	messageSize	message

Si le client refuse la connexion, un message au format suivant sera envoyé

Byte	Int	String
6	senderPseudoSize	senderPseudo

contenant le pseudo de la personne ayant refusé la communication.

Cependant si la connexion est acceptée,
un message au format suivant est envoyé

Byte	Int	String	Int	String
7	senderPseudoSize	senderPseudo		

ce message contient le pseudo de la personne

et la demande d'ouverture de connexion privé se fait au format suivant :

Byte	Int	String	Int	String
8	senderPseudoSize	senderPseudo	addressSize	address

qui contient en plus du pseudo de la personne avec qui on souhaite se connecter son addresse

Byte	Int	String	Int	String
9	senderPseudoSize	senderPseudo	addressSize	address

Finalement lorsqu'une erreur est rencontrée par le serveur, une trame : à tous les client

1 Byte
104

9. Utilisation concrète

Un serveur est démarré sur un port donné avec la commande :
java fr.upem.net.chatos.serverSide port
Les clients se connectent ensuite au serveur avec la commande :
java fr.upem.net.chatos.clientSide host port login
Ils y précisent par la même occasion l'identifiant qu'ils souhaitent prendre.
Ensuite différentes commandes peuvent être entrées :
@login message
Pour envoyer un message privé à l'utilisateur login et
@ ou / message
Pour envoyer un message public,
- login
Pour refuser une connexion privée avec l'utilisateur login,
+ login
Pour accepter une connexion privée avec l'utilisateur login.

10. Conclusion

Ce projet met à profit ce qu'on a pu voir au cours des différents tp de programmation réseau, on a dû créer des clients qui étaient capable de se connecter, de recevoir et d'envoyer des informations, on a du utiliser un système de sélecteurs et de clés pour déterminer justement si ils devaient lire ou écrire des données, on a du utiliser un format de données précis pour que les clients et le serveurs soient capable de se comprendre, en résumé réutiliser quasiment toutes les notions que nous avons vu pour la création en java de réseaux tcp.

Ce projet a aussi fait appel aux compétences développées lors du premier semestre en design pattern et java avancé car nous avons dû organiser correctement le code afin de faire en sorte qu'il soit clair et que tous les appels soient logiques, nous avons dû utiliser des formats de données vu lors du premiers semestre.