Université Gustave Eiffel AOUDIA Hakim
Request for Comments TRAN Sylvain
Category: Informational 29 Mars 2024

Chadow – Serveur de discussion en ligne

Table des matières

1. Introduction

1.1 Objectif

Le protocole décrit dans cette RFC permet de réaliser un protocole de serveur de discussion et de partage de fichier en ligne. Le protocole inclut 3 types de fonctionnalités pour les clients : l'envoie, l'annonce et la demande d'informations au serveur.

Les clients peuvent envoyer des messages à un autre utilisateur ou bien à tous les clients connectés au réseau. Pour connaître le pseudo des clients connectés au réseau, un client peut demander au serveur la liste des pseudos.

Les clients peuvent aussi annoncer le partage ou l'arrêt de partage d'un ou plusieurs fichiers. Les autres clients peuvent demander au serveur de télécharger ces fichiers mis à disposition. Pour télécharger un fichier, un client peut choisir entre le mode ouvert et le mode caché.

1.2 Définitions

proxy

Un proxy est ici un relais qui fait le lien entre un client A et un autre client B. Il relait les requêtes de A vers B et les réponses de B à A. Il est possible de chaîner plusieurs proxys.

client

Programme qui envoie des requêtes à un serveur.

serveur

Programme qui reçoit des requêtes d'un ou plusieurs clients et renvoie les réponses appropriées.

port

Un entier valant entre 0 et 65535.
exemple: 7777

adresse IP

Numéro unique identifiant une machine connecté sur un réseau utilisant IP.
exemple : adresse Ipv4 : 172.16.254.1
adresse Ipv6 : 00:00:00:00:00:00:00:00

adresse d'écoute

Adresse composée de adresse IP + port, que les clients communiqueront au serveur et où ils devront accepter des connexions TCP. Le serveur disposera également en plus de son adresse publique une adresse d'écoute qui lui servira à jouer un rôle de proxy.

pseudo

Suite de caractères choisis par un client servant à l'identifier de manière unique.

2. Fonctionnalités

Convention :

STRING(<=N) : représente une chaîne de caractères
N = limite d'octets de la chaine, str ne peux pas faire plus de N octets.
N <= 1024

```
   int   bytes
+------+-----+
| size | str |
+------+-----+
```

size : Un int (en Big Endian) donnant la taille
en nombre d'octets de la chaîne, encodée en UTF-8.
str : Les octets de la chaîne, encodée en UTF-8

IP_ADDRESS : représente une adresse IP
Code:
IPV4 = 4
IPV6 = 6

```
  byte   bytes
+--------------+
| Code | bytes |
+------+-------+
```

bytes : les 4 octets de l'adresse IPv4 si Code = 4 ou les 16 octets de l'adresse IPv6 si Code = 6

LISTENING_ADDRESS : représente une adresse d'écoute

```
 IP_ADDRESS   int
+-----------+------+
|     ip    | port |
+-----------+------+
```

port : Un int (en Big Endian) représentant un numéro de port

0. Connexion
La première connexion d'un client au serveur.

Son but est de se connecter au serveur en indiquant son pseudo + son adresse d'écoute.
Client -> Serveur :

TRY_CONNECTION = 0

```
byte  STRING(<=30)   LISTENING_ADDRESS
+---+--------------+---------+
| 0 | pseudo       | address |
```

```
    +---+--------------+---------+
```

- Côté serveur, la réponse au client :
La réponse au client sera différente selon si le pseudo est déjà occupé par un client connecté au serveur.

2 type de CODE pour la réponse :
    PSEUDO_VALIDATED = 1
    PSEUDO_REFUSED = 2

```
      byte
    +------+
    | CODE |
    +------+
```

Si le code est 0, le client doit renvoyer sa demande de connexion avec un pseudo différent.
Tant que le code est 0 le client ne peut pas utiliser le serveur de discussion en ligne.

1. Envoi de message

1.1 À tous les clients

Envoyer des messages qui seront transmis à tous les clients connectés

Client -> Serveur :

   MSG_TO_ALL = 3

```
    byte  STRING(<=1024)
   +----+---------+
   | 3  | message |
   +----+---------+
```

- Côté serveur, il récupère le message qu'il va transmettre à tous les autres clients et en précisant le pseudonyme du client émetteur dans son paquet :

   MSG_RECEIVED = 4

```
    byte  STRING(<=30)  STRING(<=1024)
   +----+------------+---------+
   | 4  |  pseudoSrc | message |
   +----+------------+---------+
```
   avec pseudoSrc le pseudo du client émetteur.

1.2 À un seul autre client

Envoyer des messages qui seront destinés à un seul client identifié par son pseudonyme.

Client -> Serveur :

   MSG_TO_ONE = 5

```
    byte  STRING(<=30)  STRING(<=1024)
   +----+-----------+---------+
   | 5  | pseudoDst | message |
   +----+-----------+---------+
```

avec pseudoDst le pseudo du client destinataire du message.

- Côté serveur,

- si le pseudo du client destinataire n'a pas été trouvé/n'est utilisé par aucun client, le serveur envoie le pseudo erroné ainsi que le message qui n'a donc pas trouvé son destinataire dans la trame suivante :

```
    PSEUDO_NOT_FOUND = 6

    byte  STRING(<=30)  STRING(<=1024)
   +----+-----------+---------+
   | 6  | pseudoDst | message |
   +----+-----------+---------+
```

avec pseudoDst le pseudo non trouvé.

- sinon, le serveur transmet le message au client destinataire dans un paquet comme ci-dessous :

```
    MSG_RECEIVED = 4

    byte  STRING(<=30)  STRING(<=1024)
   +----+-----------+---------+
   | 4  | pseudoSrc | message |
   +----+-----------+---------+
```

avec pseudoSrc le pseudo du client émetteur.

et il envoie une trame au client émetteur pour confirmer l'envoie du message :

```
    PSEUDO_FOUND = 7

    byte
   +---+
   | 7 |
   +---+
```

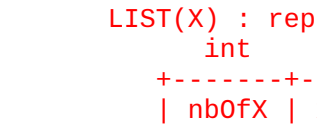
2. Annonce au serveur

Convention :

```
FILE : représente un fichier
  STRING(<=100)   bytes   int
+---------------+------+------+
| filename      | Id   | size |
+---------------+------+------+
```

filename : une STRING donnant le nom du fichier (nom + extension)
Id : un identifiant sur 16 octets calculé avec l'algorithme MD5
size : Un int (en Big Endian) donnant la taille en octet du fichier

LIST(X) : représente une liste d'élément de type X

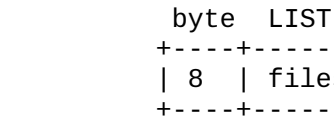


nbOfX : Un int (en Big Endian) donnant le nombre d'élément de type X

2.1 Proposition de fichiers

Le client peut annoncer au serveur qu'il propose un ou des fichiers pour le téléchargement par les autres clients.

Client -> Serveur :
SHARE_FILES = 8



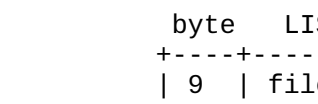
avec filesShare la liste des fichiers partagés par le client.

- Le serveur se contente de stocker la liste des fichiers partagés pour ce client spécifiquement.

2.2 L'arrêt de partage de fichiers

Le client peut annoncer au serveur qu'il ne propose plus un ou des fichiers pour le téléchargement par les autres clients.

Client -> Serveur :
STOP_SHARE_FILES = 9



avec filesUnshare la liste des fichiers que le client ne veut plus partager.

- Le serveur se contente de retirer les fichiers qui ne sont plus partagé par le client.

3. Demande au serveur

3.1 Liste pseudos

Le client peut demander au serveur la liste des pseudo des clients connectés au serveur.

Le client envoie une trame au serveur contenant un code permettant au serveur de savoir que le client demande la liste des pseudos.

Client -> Serveur :
REQUEST_LIST_AVAILABLE_PSEUDOS = 10

```
     byte
    +----+
    | 10 |
    +----+
```

- Côté serveur, il va transmettre le nom des pseudonymes dans une trame comme ci-dessous :

```
    AVAILABLE_PSEUDOS = 11

    byte  LIST(STRING)
   +----+---------+
   | 11 | pseudos |
   +----+---------+
```

3.2 Liste fichiers

3.2.1 Partagés par tout les clients

Le client peut demander au serveur la liste des fichiers disponibles au téléchargement.

Le client envoie une trame au serveur contenant un code permettant au serveur de savoir que le client demande la liste des fichiers disponibles au téléchargement.

Client -> Serveur :

```
    REQUEST_LIST_AVAILABLE_FILES = 12
     byte
    +----+
    | 12 |
    +----+
```

- Côté serveur, il va transmettre la liste de tous les FILE disponibles sans répétitions dans une trame comme ci-dessous :

Serveur -> Client :

```
    AVAILABLE_FILES = 13

    byte  LIST(FILE)
   +----+-------+
   | 13 | files |
   +----+-------+
```

3.2.1 Partagés par lui même

Le client peut demander au serveur la liste des fichiers qu'il a déjà partagé au serveur.

Le client envoie une trame au serveur contenant un code permettant au serveur de savoir que le client demande la liste des fichiers qu'il avait partagé précédemment.

Client -> Serveur :

```
    REQUEST_LIST_FILES_SHARED_BY_ME = 28
     byte
    +----+
    | 28 |
    +----+
```

- Côté serveur, il va transmettre la liste de tous les fichiers que le client partage :

Serveur -> Client :
    FILES_SHARED_BY_ME = 29

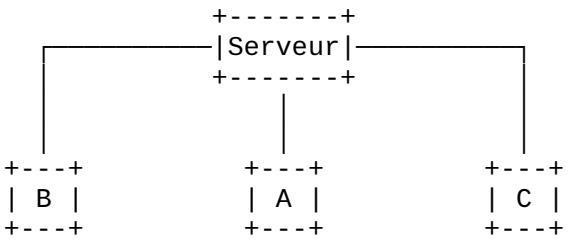
    byte LIST(FILE)
   +----+-------+
   | 29 | files |
   +----+-------+

3.3 Télécharger fichier

Un client peut demander à télécharger un fichier soit en mode ouvert, soit en mode caché :

Légende :
    Clients connectés : A, B, C
    ——————————————— : Connexion avec le serveur
    X ------------> Y : Envoie d'une trame d'un client X à un client Y

   Exemple d'un état d'un réseau où les clients A, B et C sont connectés au serveur :



Convention :

CHUNK : représente un morceau de fichier.
   int   bytes
+------+---------+
| size | payload |
+------+---------+

size : Un int (en Big Endian) entre 1 et 1024 donnant la taille du morceau
payload : les size octets du morceau

-Un fichier est découpé en morceaux qui sont numérotés de 0 à ceil(SIZE/1024)-1

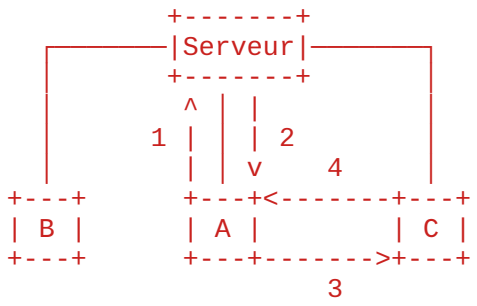
où ceil(x) est la méthode qui calcule l'arrondi entier supérieur de x et SIZE = le int (en Big Endian) donnant la taille en octet du fichier

Tous les morceaux font 1024 octets sauf le dernier qui peut faire moins de 1024 octets.

ChunkNumber : Un int (en Big Endian) donnant le numéro d'un chunk.

1. Mode ouvert :

Schéma exemple :



Pour télécharger un fichier en mode ouvert :

1 – le client A demande au serveur la liste des adresses d'écoute des clients partageant ce fichier :

REQUEST_LIST_SHARER = 14

byte	FILE
14	file

2 – Le serveur répond avec la liste de toutes les adresses d'écoute des clients partageant le fichier :

LIST_SHARER = 15

byte	LIST(LISTENING_ADDRES)
15	list

A pourra ensuite se connecter à ces adresses et demander des morceaux du fichier.

3 – Le client initiant A peut demander le morceau numéro chunkNumber du fichier avec la trame suivante :

OPEN_REQUEST = 16

byte	FILE	int
16	file	chunkNumber

4 – Le client acceptant, ici C, à 2 choix, soit il accepte, soit il refuse de partager son chunk :

-S'il refuse, il enverra la trame :

OPEN_REQUEST_DENIED = 17

byte	FILE	int
17	file	chunkNumber

-S'il accepte, il enverra la trame:

OPEN_RESPONSE = 18

byte	FILE	int	CHUNK
18	file	chunkNumber	chunk

où file et chunkNumer sont les mêmes que la OPEN_REQUEST à laquelle C répond.

2. Mode caché:

Un **token** est un int (en Big Endiant).

Chaque client devra maintenir:
- une table de routage qui associe à des token unique une paire (LISTENING_ADDRESS, token)
- une liste de tokens terminaux

Convention :
TOKEN = représente un token

Pour télécharger un fichier en mode caché :

1 – Le client demande au serveur une liste de paires d'adresse d'écoute et de token par lesquels télécharger le fichier

REQUEST_LIST_PROXY = 19

byte	FILE
19	file

2 – Le serveur indique aux clients jouant un rôle de proxy comment remplir leur table de routage :

PROXY = 20

byte	TOKEN	LISTENING_ADDRESS	TOKEN
20	tokenSrc	dst	tokenDst

3 – le(s) client(s) qui reçoit la trame ajoute la paire (dst, tokenDst) pour le token tokenSrc dans sa table et envoie la trame suivante au serveur pour signaler que sa table a été mise à jour :

PROXY_OK = 21

byte	TOKEN
21	tokenSrc

4 – Le serveur envoie la trame suivante au(x) client(s) possédant le fichier à télécharger afin qu'il remplit sa liste de tokens terminaux :

TERMINAL = 22

byte	TOKEN
22	tokenSrc

5 – le(s) client(s) qui reçoit la trame ajoute le token tokenSrc dans sa liste de tokens terminaux et envoie la trame suivante au serveur pour signaler que sa liste a été mise à jour :

TERMINAL_OK = 23

byte	TOKEN
23	tokenSrc

6 – Après s'être assuré que toutes les mises à jour ont bien été effectuées, c'est-à-dire après avoir reçu toutes les réponses PROXY_OK et TERMINAL_OK
, le serveur répond à la trame de l'étape 1 par une liste d'adresses d'écoute et de tokens :

LIST_PROXY = 24

byte	LIST((LISTENING_ADDRES, TOKEN))
24	list

Le client pourra ensuite se connecter à ces adresses et demander des morceaux du fichier.

7 – Le client initiant peut demander le morceau numéro chunkNumber du fichier avec la trame suivante :

HIDEN_REQUEST = 25

byte	FILE	int	TOKEN

| 25 | file | chunkNumber | token |
+----+------+-------------+-------+

où token est le token associé à l'adresse d'écoute du client acceptant auquel on fait la requête.

Lorsqu'un client B reçoit une HIDEN_REQUEST avec un token t d'un client A, il y a 3 cas possibles :

Cas 1 - t n'est ni dans la table de routage, ni dans l'ensemble des tokens terminaux, alors le client B ferme la connexion avec le client A

Cas 2 - t est dans la table de routage associée à l'adresse d'écoute d'un client C avec un token t'. Dans ce cas, le client B transfère la requête au client C en remplaçant le token t par t'

Cas 3 - t est dans l'ensemble des tokens terminaux, alors le client B à 2 choix, soit il accepte, soit il refuse de transmettre le morceau.

-S'il refuse, il enverra la trame :

HIDEN_REQUEST_DENIED = 26

byte	FILE	int	TOKEN
26	file	chunkNumber	token

-S'il accepte, il enverra la trame:

HIDEN_RESPONSE = 27

byte	FILE	int	CHUNK	TOKEN
27	file	chunkNumber	chunk	token

où file, chunkNumber et token sont les mêmes que la HIDEN_REQUEST à laquelle le client acceptant B répond.

Quand un client B reçoit une HIDEN_RESPONSE ou une HIDEN_RESPONSE_DENIED avec un token t, il y a trois cas :

Cas 1 - cette réponse correspond à une question qu'il a posée pour un téléchargement. Dans ce cas, le client a sa réponse.

Cas 2 - ce token n’apparaît pas associé à un autre token t' dans la table de routage, le client ferme la connexion.

Cas 3 - ce token apparaît associé à un unique autre token t' dans la table de routage. Le client transmet la réponse en remplaçant le token t par le token t' sur la connexion sur laquelle il a reçu la requête HIDEN_REQUEST correspondante avec le token t'.