RFC Chadow version 1.0.0

Mars 2024

Valentin Sebbah
Antoine Bastos

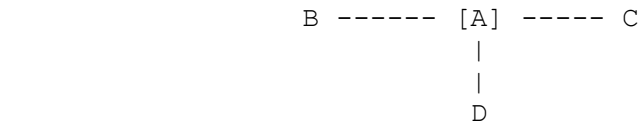
DESCRIPTION

Le protocole Chadow est un protocole d'échange de message et de fichiers. Il permet la mise en place d'un reseau de partage décentralisé de fichiers entre clients. Il embarque un moyen d'éventuellement anonymiser le téléchargement en passant par d'autres clients qui servent de proxy.

SOMMAIRE

1. Conventions

Il existe des noeuds:
- serveur, représenté par une lettre majuscule entre crochet, ex:[A]
- client, représenté par une lettre majuscule, ex: B



Chadow utilise le protocole TCP (Transmission Control Protocol) comme fondation pour assurer la fiabilité et l'intégrité des communications entre les clients et le serveur.

Codex : Une suite de données décrivant les fichiers partagés par un ou plusieurs utilisateurs dans un réseau pair à pair. Chaque codex est identifié par un identifiant unique (ID) et comprend des informations telles que la liste des fichiers partagés, les parties de ces fichiers disponibles chez chaque utilisateur (sharer), et d'autres métadonnées pertinentes.

Annuaire : Un serveur centralisé qui stocke une liste de codex disponibles dans le réseau. L'annuaire permet à tout client de récupérer les informations nécessaires sur les codex afin de participer au partage de fichiers. En consultant l'annuaire, un client peut découvrir quels fichiers sont disponibles pour le téléchargement, ainsi que les utilisateurs (sharers) qui possèdent des parties de ces fichiers.

Client : Une application utilisée pour télécharger et partager des fichiers dans un réseau pair à pair. Les clients interagissent avec l'annuaire pour obtenir des informations sur les fichiers disponibles et établissent des connexions avec d'autres utilisateurs (sharers) pour échanger des parties de fichiers.

Sharer : Dans le contexte d'un réseau de partage de fichiers pair à pair, les sharers sont les utilisateurs qui possèdent et partagent des parties de fichiers avec d'autres utilisateurs. Chaque sharer contribue au réseau en mettant à disposition des parties de fichiers qu'il possède, permettant ainsi aux autres utilisateurs de télécharger ces fichiers à partir de plusieurs sources.

2. Encodage

Opcode des trames

0	error	1	register
2	discovery	3	event

4	yell	5	whisper
6	propose	7	request
8	announce	9	update
10	request	11	handshake
12	keepalive	13	choke
14	unchoke	15	interested
16	notinterested	17	have
18	needblock	19	cancel
20	hereblock	21	newchunk
22	proxyopen	23	anonymize
24	disconnect		

- un int représente 4 octets
- un long représente 8 octets
- une string est toujours précédée par sa taille en int (4 octets) et encodé en UTF-8

size (int)	mystring (string)

- 'creation_date' est une date au format UNIX (long), nombre de secondes depuis le 1er janvier 1970 à 00:00:00 UTC

- 'username' et 'username_receiver' se réfèrent à une chaine de charactères (string):
  - D'une taille comprise entre 2 et 32 charactères
  - Encodée en UTF-16 (Unicode)
  - Ne contenant pas d'espaces ou de charactères invisibles
  - Ne contenant pas les sous-chaînes suivantes : @, #, :, /, ```, chadow

  Le serveur doit garantir qu'il n'existe pas deux clients avec le même username.

3. Identification

Le client doit connaître l'adresse du serveur avec lequel il souhaite communiquer.

Lorsqu'un client souhaite se connecter au serveur IRC, il envoie une demande de connexion en spécifiant le nom d'utilisateur qu'il souhaite

utiliser. Voici le format de la trame pour cette demande :

```
+-----------------------+
| (register) | username |
+-----------------------+
```

Le serveur répond au client pour indiquer si le nom d'utilisateur est disponible ou non :

- Si le nom d'utilisateur est disponible, le serveur envoie une validation au client :

```
+-----------------------+
| (register) | username |
+-----------------------+
```

- Si le nom d'utilisateur est déjà utilisé ou que le nom d'utilisateur ne respecte pas les règles de formatage définies dans la section "Encodage", le serveur envoie un message d'erreur au client :

```
+----------------------------------+
| (error) | error_message (string) |
+----------------------------------+

error_message: message human-readable
```

Dans ce cas ci le client peut envoyer à nouveau une demande de connexion au serveur.

4. Discussion

4.1 Découverte des utilisateurs

Le client envoie au serveur la trame suivante:

```
+-------------+
| (discovery) |
+-------------+
```

Le serveur répond avec la liste des noms d'utilisateurs présents:

```
+-------------------------+
| (discovery) | usernames |
+-------------------------+
```

usernames:
liste des usernames des clients connectés. Concaténation des utilisateurs <taille de l'username en int><username>.
La liste se termine par 0.
Ex: 8morpheus7trinity0

4.2 Mise à jour des présences

Le serveur notifie les évènements ('code') d'arrivé (1) et de départ

(0) à tous les clients. Permet d'éviter de demander la liste entière des utilisateurs systématiquement.
Finalement les arrivés et départs sont traités comme des messages globaux.

(event)	code (byte)	username

username: le username associé avec l'évènement

4.3 Publication de messages globaux

Le client envoie une trame de publication de message global.

(yell)	creation_date	message (string)

Le serveur retransmet la même trame à tous les clients même le client d'origine.

4.4 Echange de messages entre utilisateurs

Le client envoie au serveur une trame pour un message privé destiné à un username spécifique.

(whisper)	username_receiver	creation_date	message (string)

Le serveur la retransmet au client associé.

Si le serveur ne connaît pas l'username (n'est plus connecté ou n'existe pas), le serveur réponds une trame d'erreur.

(error)	error_message (string)

message: description de l'erreur human-readable

5. Codex

L'annuaire possède des descriptions de chaque partage qu'on appelle codex.
L'annuaire sait quels clients partage un codex.
Les clients peuvent chercher des codex dans l'annuaire.

Les clients peuvent demander à l'annuaire de rejoindre le réseau de partage des fichiers du codex.

5.1 Publication du codex

L'intérêt est que le l'annuaire ne possède pas de copie des fichiers en partage, ce qui reviendrait à du téléchargement centralisé et nécessite un serveur très puissant.
L'annuaire possède seulement une description des fichiers que les sharers partagent.

Le client peut proposer un ou des fichiers au partage qui sont rassemblés dans un même ensemble décrit pas le codex.

```
+------------------------------+
| (propose) | codex_id | codex |
+------------------------------+
```

codex_id : le hachage (SHA1) du codex.
codex : une structure comprenant les informations suivantes :
- name : le nom du codex (string).
- chunk_length : la taille fixe des chunks du codex.
- files_count : le nombre de fichiers (int).
- chunks : la concaténation des identifiants (SHA1) de chaque chunk.

Chaque fichier du codex est découpé en chunk d'une taille fixe en int, d'une puissance 2. De cette façon plusieurs sharers peuvent partager des bouts du fichiers en simultané, ce qui augmente sa disponibilité et la probabilité qu'un client aboutisse à compléter le téléchargement. Dans un optique de pouvoir télécharger des fichiers d'une taille jusqu'à 2Go on propose une taille de 256ko qui donne un total de 7 813 chunks.
Chaque chunk est ensuite hashé pour obtenir un identifiant SHA1. Pour un fichier de 2Go cette liste de sha1 (20 bytes) réprésente un poids de 156ko, ce qui est raisonnable. Le hashage permet aux clients de s'assurer de l'intégrité des données qu'il télécharge, en testant le hash du chunk reçu avec celui décrit dans le codex.

files:
une liste qui concatène tous les <nomfichier><path> des fichiers du codex. nomfichier et path sont des strings.
path représente le chemin absolu où nomfichier se situe dans le codex.

5.2 Téléchargement du codex

Le client peut télécharger le codex avec la trame:

```
+------------------------------+
| (request) | codex_id (SHA1) |
+------------------------------+
```


Le serveur répond avec la trame suivante :

```
+-------------------+
| (request) | codex |
+-------------------+
```

où codex contient les mêmes informations que décrites dans la partie 5.1.

6. Recherche de codex

Le client envoie une trame de recherche au serveur avec les informations suivantes :

(search)	codex_name	options	expected_results	creation_date

codex_name : (string) le nom du codex à rechercher.
options : un champ de bits (int) où chaque bit est numéroté à partir du poids fort.
- Le bit 1 indique une recherche à une date spécifique (incluse).
- Le bit 2 indique une recherche avant une certaine date (exclue).
- Le bit 3 indique une recherche après une certaine date (exclue).

expected_results : (int) le nombre de résultats attendus, avec 0 pour "tous".

exemple de trame pour le client qui recherche le codex "MonCodex":

(search)	MonCodex	2	5	1646232387

Le serveur répond avec les informations suivantes :

(search)	number_of_results	results

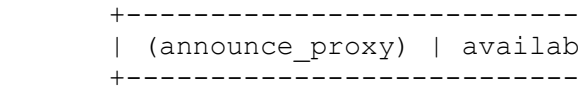
number_of_results : (int) le nombre total de résultats.
results : une liste comprenant :
- filename : (string) le nom du fichier.
- codex_id : (20 bytes) le hash d'information.
- creation_date : (long) la date de création.
- seeders : (int) le nombre de seeders (sharers qui partagent et possèdent les fichiers du codex à 100%).
- leechers : (int) le nombre de leechers (clients qui partagent les fichiers du codex mais qui ne les ont qu'en partie).

7. Annonce

Le client annonce à intervalle régulier, défini par le serveur, son état pour chaque codex partagés ou téléchargés, ainsi que sa disponibilité à servir de proxy.

7.1 Annonce de proxy

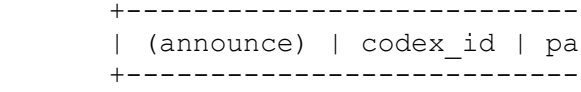
Le client envoie une trame pour annoncer s'il est disponible ou non pour servir de proxy :



availability: (byte)
- 0 : indique que le client n'est pas disponible pour servir de proxy.
- 1 : indique que le client est disponible pour servir de proxy.

Le client effectue cette annonce à chaque changement d'état et doit renvoyer une annonce une minute après la dernière annonce pour maintenir sa disponibilité actuelle à jour.

7.2 Annonce de partage ou de téléchargement



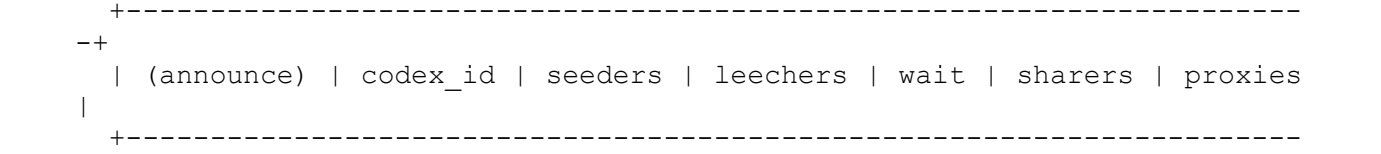
codex_id: (SHA1) hash du dictionnaire de données associé au partage
paramètres: une liste comprenant :
- event: (byte)
  - 0 : indique que le client est en train de télécharger le codex.
  - 1 : signifie que le client a cessé de partager le codex.
  - 2 : indique que le client a complètement téléchargé le codex.

  Ce paramètre spécifie l'état actuel du codex.
- privacy: (byte)
  - 0 : signifie que le partage est en mode caché.
  - 1 : indique que le partage est en mode ouvert.

  Ce paramètre détermine la visibilité du partage.
- sharers:

  (int) le nombre de sharers que le client souhaite recevoir du serveur pour garantir le téléchargement du codex.

Le serveur répond avec une trame pour répondre à la requête :



seeders: le nombre de sharers ayant les codex complet (int)
leechers: le nombre de clients ayant en partie les codex (int)
wait: l'intervalle minimum que le client doit attendre avant de refaire une annonce (int)

sharers: une liste de sharers actifs au partage du codex, chaque
sharer étant représenté par une série de 6 octets
d'informations:

```
+-----------------------------------------------+
| sharers_count | ip | port | ip | port | ... |
+-----------------------------------------------+
```

- sharers_count: le nombre de sharers actifs (int)
- ip: l'adresse IP du sharer (4 octets)
- port: le port du sharer (2 octets)

proxies: présent lorsque le client télécharge en mode fermé.
Liste des clients disponibles pour agir en tant que proxy :

```
+-----------------------------------------------+
| proxies_count | ip | port | ip | port | ... |
+-----------------------------------------------+
```

- proxies_count: le nombre de clients disponibles pour agir en tant
que proxy (int)
- ip: l'adresse IP du client (4 octets)
- port: le port du client (2 octets)

En cas d'erreur, le serveur répond avec la trame suivante :

```
+----------------------------------+
| (error) | error_message (string) |
+----------------------------------+
```

error_message: message human-readable
Le client pourra alors renvoyer une annonce

8. Téléchargement des fichiers du codex

8.1 Mode ouverte

A ce stade le client possède un codex et donc
connaît la liste des chunks du codex et les fichiers qu'il contient.
Il connaît également les sharers qu'il doit contacter pour récupérer
les chunks.

Le client initie l'échange avec un sharer avec le handshake:

```
+-------------------------+
| (handshake) | codex_id  |
+-------------------------+
```


Si le client reçoit un handshake avec un codex_id qu'il ne possède
pas où qu'il ne souhaite pas partager il peut terminer la connexion.
Dans ce cas on

Le reste des trames échangées est préfixées par leur taille, afin de

facilter la lecture sur le socket par le client.

8.1.1 Champs de bits

Juste après le handshake le client et le sharers s'envoient respectivement le champs de bits des chunks qu'ils possèdent du codex.
Le bit de poids fort est le chunk d'indice 0.
Le client termine la connexion si le champs de bits n'est pas de la bonne taille.

```
+-------------------------------------+
| (have) | bitfield_length | bitfield |
+-------------------------------------+
```

bitfield_length: Longueur du champ de bits (int).
bitfield: Champ de bits représentant les chunks possédés par le client ou le pair.
Chaque bit correspond à un chunk du codex, où le bit de poids fort représente le
chunk d'indice 0. Un bit à 1 indique que le chunk correspondant est possédé, tandis qu'un bit à 0 indique qu'il ne l'est pas.

Le client connaît les chunks que possèdent les sharers avec lesquels il est connecté grâce aux bitfields échangés.
Lorsqu'il reçoit un nouveau chunk, le client sait quels sharers ne le possèdent pas.
Il envoie alors une trame newchunk à ces sharers. Ces derniers pourrons alors demander ce chunk au client.

```
+--------------------------+
| (newchunk) | chunk_index |
+--------------------------+
```

Cas de figure

```
                    A ---- C ---- B
```

Dans cette configuration à l'ouverture des deux connexions:
- A connaît les chunks de C
- C connaît les chunks de A et B
- B connaît les chunks C

Au bout d'un moment C complète un chunk qu'il a téléchargé chez B.
C envoie (newchunk)<chunk_index> à A.
A qui n'avait pas le chunk et ne pouvait pas le télécharger chez C, peut désormais le faire.

8.1.2 Keep-alive

Le client doit aussi maintenir la connexion avec des message keep-alive, dans le cas contraire le pair peut y mettre fin s'il n'a rien reçu du client pendant une certaine durée.
La durée définie est de une minute et 30 secondes.

```
+-------------+
| (keepalive) |
+-------------+
```

8.1.3 Demande de block

Le client souhaite télécharger un chunk au sharer.
Le client ne télécharge pas les chunks mais par portions
de chunks appelés 'blocks' de taille comprise entre 16ko et la
taille du chunk.

```
+-----------------------------------------------+
| (needblock) | chunk_index | offset | length |
+-----------------------------------------------+
```

chunk_index: l'indice du chunk demandé (int).
offset: l'indice du byte dans le chunk (int).
length: la taille du block (int).

8.1.4 Annulation

Le client peut envoyer une trame pour annuler la requête
(need_block) d'un block. La trame est identique à celle envoyée
pour la requête mais l'opcode est remplacé par (cancel).

8.1.5 Envoie du block

Le sharer répond à une requête de block:

```
+-----------------------------------------------+
| (hereblock) | chunk_index | offset | payload |
+-----------------------------------------------+
```

chunk_index: l'indice du chunk demandé (int).
offset: l'indice du byte dans le chunk (int).
payload: la donnée

8.2 Mode fermé

À ce stade, le client a envoyé une demande au serveur pour télécharger
un codex en mode fermé. Si le nombre de clients est insuffisant pour
garantir le téléchargement du codex (seulement 2 client sont présent
sur le serveur), le serveur agira comme un proxy pour le client,

échangeant ainsi les trames de demande de bloc avec le client et les
trames de réponse avec le sharer.

8.2.1 Connexion en chaîne

Après avoir reçu une liste de proxy par le serveur après
son l'annonce, le client se connecte à un des proxys.

```
+-------------------------------------------------+
| (proxyopen) | codex_id | download_id | chain_id |
+-------------------------------------------------+
| remaining_proxy_count                           |
+-------------------------------------------------+
```

codex_id : hachage (SHA1) du codex.
download_id : identifiant unique du téléchargement (long)
généré aléatoirement par le client.
chain_id : identifiant unique de la chaîne de proxy (long)
généré aléatoirement par le client.
remaining_proxy_count : nombre de proxy restants dans la chaîne (int).

Après avoir reçu la trame et établi la connexion, le proxy envoie une trame au serveur pour demander une adresse pour le prochain proxy de la chaîne :

```
+-------------------------------------------------+
| (anonymize) | codex_id | download_id | chain_id |
+-------------------------------------------------+
| remaining_proxy_count                           |
+-------------------------------------------------+
```

Le serveur répond:

```
+-------------------------------------------------+
| (anonymize) | codex_id | download_id | chain_id |
+-------------------------------------------------+
| exists (byte) | ip | port                       |
+-------------------------------------------------+
```

exists:
- 0 : le serveur n'a pas pu fournir de nouveau proxy.
- 1 : le serveur a fourni une adresse pour le prochain proxy.

ip : adresse IP du client servant de proxy (4 octets).
port : port du client servant de proxy (2 octets).

Lorsque remaining_proxy_count est à 0, le serveur envoie une trame de réponse au dernier proxy de la chaîne, contenant les informations du sharer dans ip et port.
Il associe le codex_id, l'id de téléchargement, l'id de chaînage le client qu'il lui à transmis pour ne pas le lui transmettre à nouveau pour une nouvelle chaîne.

Le client peut alors utiliser les mêmes trames que dans le mode ouvert pour télécharger les chunks, les proxy servant de relais pour les trames de demande de bloc et de réponse.

8.2.2 Réparation de la chaîne

A chaque étape de la création du chaînage, le maillon (client ou proxy) peut abandonner le chaînage s'il n'a pu établir de connexion

avec le prochain maillon après écoulement d'un timeout qu'il aura choisi (l'annuaire ne peut pas fournir de nouveau proxy ou de sharer, problème de latence,...).
Les proxys en amont peuvent décider de suivre ou de retenter la construction de la chaîne.
En cas de timeout non écoulé, le proxy demande en boucle à intervalle de 4 secondes le nouveau maillon (proxy ou sharer).

```
(Left)                                  (Right)
Client ---> Proxy A ---> Proxy B ---> Sharer
```

Toute fermeture de connexion prevenant du côté gauche (côté client) d'un maillon de la chaîne entraîne la fermeture successive des connections de la chaîne à droite (côté sharer).
Toute fermeture à droite d'un maillon de la chaîne entraîne le même comportement que l'établissement de la chaîne décrit plus haut.
Ce fonctionnement optimiste permet la reconstruction d'un chaîne rompue en considérant que les proxy et l'état de partage du codex seront favorables.

8.2.3 Déconnexion en chaîne

Dès qu'un proxy, le client ou le sharer se déconnecte, une déconnexion en chaîne est initiée. Le client doit informer le serveur qu'une chaîne a été fermée, lorsque que la connection avec le premier proxy de la chaîne a été fermée.
Le client envoie alors une trame de déconnexion au serveur :

(disconnect)	codex_id	download_id	chain_id

9. Deconnexion

Lorsqu'un client se déconnecte du serveur, que ce soit de manière inattendue ou volontaire, le protocole TCP permet au serveur de détecter automatiquement cette déconnexion. Le serveur réagit à cette déconnexion en libérant les ressources associées à cet utilisateur et en informant tous les autres utilisateurs de sa déconnexion, conformément aux spécifications de la partie "Discussion - Mise à jour des présences".