Hakim AOUDIA
Sylvain TRAN

Semestre 2
Master 1 informatique

Université Gustave Eiffel

Projet de Réseau : CHADOW

Introduction

Guide utilisateur

Voir le fichier README qui indique comment construire les jars et comment les clients peuvent utiliser le protocole.

Il y a un fichier run.sh qui permet de lancer directement le serveur et 3 clients. Pour lancer le fichier il faut utiliser la commande suivante :
sh run.sh
ou
./run.sh

Ce qui fonctionne

- Les fonctionnalités clients <-> serveur sont toutes implémentées et fonctionnelles
- Téléchargement d'un petit fichier vers un seul autre client
- Téléchargement d'un gros fichier vers un seul autre client
- Téléchargement d'un petit fichier vers plusieurs autre clients
- Téléchargement d'un gros fichier vers plusieurs autre clients
- Déconnexion d'un client fournissant le fichier pendant le téléchargement même chose pour les téléchargements avec les proxys

Ce qui fonctionne pas

Tout à l'air de fonctionner a prioris.

Extrait de code des éléments demandés lors de la soutenance beta

Voici les feedbacks données :

Précisez le type d'IP utilisé car sinon un string est trop vague.

- Solution :

```
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
```

-

Préciser le protocole de cryptage utilisé

- Solution : On utilise le **MD5** pour identifier un fichier de façon unique.

Si on met un message pour dire que le message s'est mal passé il faut aussi faire un message pour dire qu'il s'est bien passé

- Solution : On a ajouté dans notre RFC un code pour indiquer si le message a été reçu ou non.

```
case Codes.PSEUDO_FOUND:
  System.out.println("Message well sent !");
  state = State.WAITING_FOR_OPCODE;
  break;
```

Mettre la taille du contenu du fichier lors du partage pour éviter problème de content

- Solution : On a modifié notre RFC pour avoir un champ ‘size’ dans un fichier.

```
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
```

```
public record FILE(String filename, ByteBuffer id, int size) implements X {
```

Donner le port du client d'écoute lors de l'authentification (au début) quand il se connecte pour éviter de demander à chaque fois.

- Solution :

```
var listeningAddress = Tools.sscToListeningAddress(listenSocket);
var bb = new TryConnection(login, listeningAddress).asBuffer();
```

```
public static LISTENING_ADDRESS sscToListeningAddress(ServerSocketChannel ssc) throws IOException {
  var address = (InetSocketAddress) ssc.getLocalAddress();
  var port = address.getPort();
  var ip = address.getAddress();

  var bytes = ByteBuffer.allocate(1_024);
  bytes.put(ip.getAddress());
  bytes.flip();

  var code = (byte) bytes.remaining();

  return new LISTENING_ADDRESS(code, bytes, port);
}
```

- **On ne peut pas faire un ServerSocketChannel à chaque fois pour un proxy, juste mettre un token d'identification pour dire qui est qui.**

```
private final HashMap<Integer, Context> mapTokenContext = new HashMap<>();
private final Set<Integer> initialTokens = new HashSet<>();
```

Ne pas parcourir tous les Context pour trouver un client

- Solution : On utilise une map.

```
// to send message to one client
private final HashMap<String, Context> loginAndContextMap = new HashMap<>();
```

```
private void sendPrivateMsgToOne(String loginDst, ByteBuffer bb) {
  var value = loginAndContextMap.get(loginDst);
  if (value != null) {
    value.queueBb(bb);
  }
}
```

Répartition des tâches

On pourrait dire 60% Sylvain TRAN et 40% Hakim AOUDIA mais on a toujours travaillé ensemble que ce soit à la fac ou bien en vocal sur discord. Pour chacun de nos commit on s'est aidé.

Points dont on est fiers

On apprécie la classe FileInformation, dans laquelle on a utilisé un BitSet pour l'efficacité. Cette classe nous permet de gérer les chunks de fichiers. Elle est utilisée pour le téléchargement de fichiers. On peut récupérer un bout d'un fichier (chunk) et aussi concerner de nombreux chunk pour former un fichier.

On a testé avec des fichiers de 10 méga octets, 1 giga et 10 gb et le protocole fonctionne avec toutes ces tailles de fichiers différents et rapidement.