Semestre 8

Avril 2022

Rapport final de projet réseaux

Filière Informatique - ENSEIRB-MATMECA

Auteurs :
CAVALAR Laszlo
DO Nicolas
KUNOW Johannes
MOUGOU Yassine
NASDAMI Quatadah
OURIHA Radouane

Encadrant :
MENDIBOURE Leo

1 Introduction

Ce document est le rapport final du projet de réseaux, portant sur le développement d'une application pour le partage de fichiers en mode pair à pair. La version implémentée correspond à la version **centralisée** où une entité centrale **Tracker** est implémentée pour assister les différents **pairs** dans la recherche des fichiers à télécharger. Les principales fonctionnalités de chacune de ces entités sont présentées dans la Figure 1.

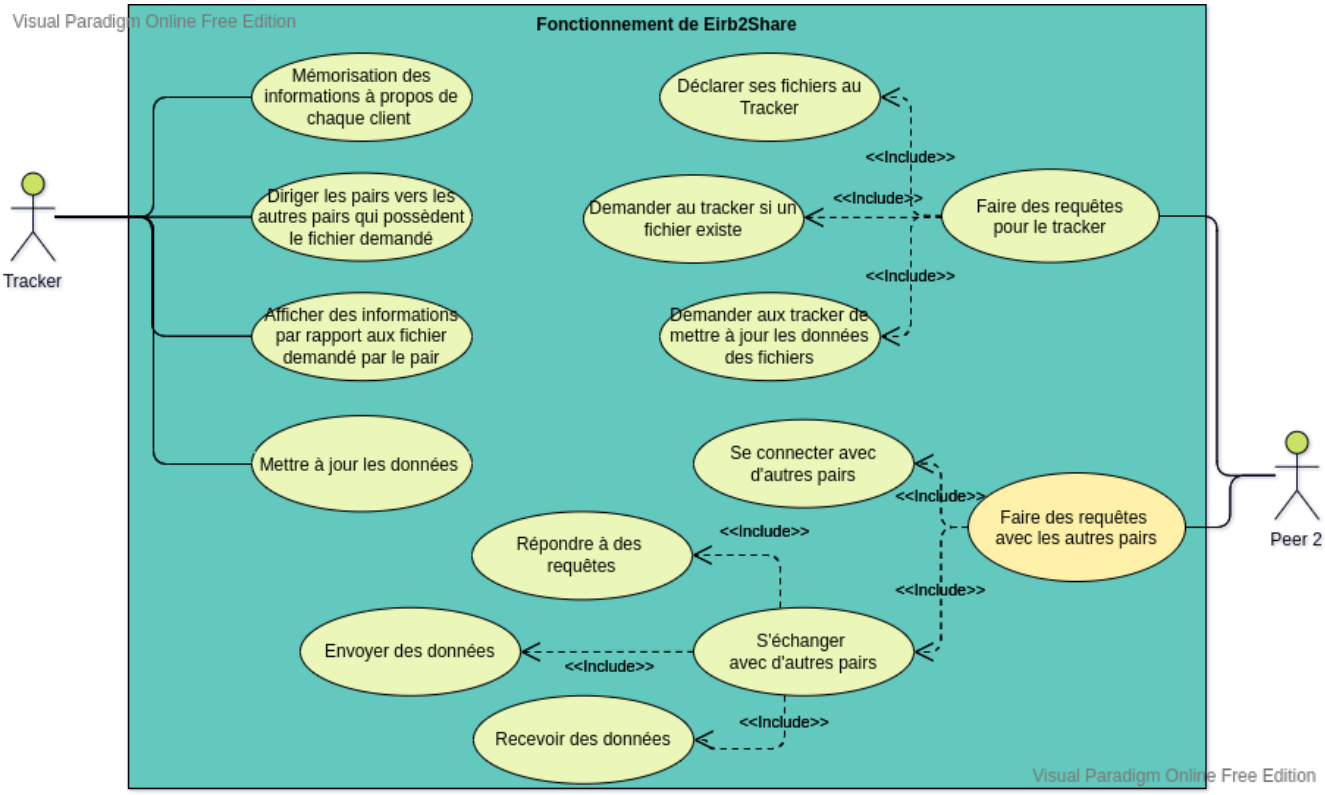

FIGURE 1 – Fonctionnalités principales

2 Organisation des projet

Le projet se décompose en deux parties. L'une qui traite le Tracker en C et l'autre les pairs en Java. Une partie de l'équipe s'est chargé de la partie Java, et l'autre partie a travaillé sur le tracker. Nous avons fait en sorte de fréquemment changé les équipes afin que tout le monde ait une connaissance globale du projet et non pas que sur une partie précise. Pour communiquer au sein du groupe, nous avons mis en place un serveur discord où chaque membre pouvait faire état de son avancement ou des problèmes qu'il rencontrait. Les principaux problèmes qu'on a pu soulever est le manque de compréhension dans le sujet, on a eu du mal à comprendre comment devait fonctionner notre application ce qui a mené à beaucoup de débats et de tests.

3 Tracker

Le rôle principal du Tracker est de centraliser les informations concernant les fichiers que possèdent les pairs. Ainsi pour tester les différentes fonctionnalités qu'offrent le Tracker, un faux client a été implémenté en **C**.

3.1 Stockage des données

Au début, chaque pair commence par annoncer sa présence et les différents fichiers qu'il possède au Tracker. Deux structures File et FileArray ont été mises en place, la premier stocke les informations de chaque fichier et l'autre représente un tableau de File gérer d'une manière dynamique.

D'autre part, pour gérer la liste des pairs nous utilisons les macros définies par la librairie "sys/queue.h". Il est aisé d'effectuer des opérations sur notre liste de pairs.

3.1.1 Peer

Un pair doit spécifier le port d'écoute utilisé par lui même et son adresse IP. On représente donc un pair avec la structure `Peer` définie comme suit :

```
struct Peer{
    int port;
    char * ip;
    FileArray * peerFiles;
    SLIST_ENTRY(PEER) next_peer;
}
```

Les attributs sont comme tels :

- `port` : port d'écoute utilisé par le pair
- `id` : l'adresse ip du pair
- `peerFiles` : un pointeur vers la liste des fichiers que possède le pair
- `next_pair` : entrée dans la liste

3.1.2 Fichier

```
struct File{
    char* file_name;
    int length;
    int piece_size;
    char* key;
}
```

Les attributs sont comme tels :

- `file_name` : le nom du fichier.
- `length` : la taille du fichier en octets.
- `piece_size` : la taille des pièces du fichier en octets.
- `key` : la clé du fichier

```
struct FileArray{
    File **array;
    size_t used;
    size_t size;
```

```
}
```

— `array` : un tableau de fichier.
— `used` : le nombre de fichier stocké dans le tableau.
— `size` : la capacité du tableau.

3.2 Implementation du Tracker

Pour gérer chaque connexion des pairs séparément, nous utilisons des threads. En effet, le thread principale du `Tracker` tourne en boucle pour accepter les connexions des pairs. Or, les threads créés gèrent la réception des messages du pair, pour cela ils disposent de la fonction `connection_handler`.

3.2.1 Parsage

Afin de parser les différents messages envoyés par le pair, le tracker doit disposer de tous les types de requêtes possibles, d'où la nécessité de créer le type énuméré suivant `REQUEST_TYPE`, ainsi qu'une fonction nommée `getRequestType` qui prend en paramètre le message et retourne le type de requête qui lui correspond. Lorsque le tracker reçoit une requête envoyée par un pair, il doit vérifier dans un premier temps que la forme du message envoyé est bien valide, en s'assurant de la valeur de chaque élément du message. Dans le cadre de notre sujet, le protocole de communication pairs-tracker est essentiellement décrit par 3 types de requêtes, une première qui a pour but d'annoncer la présence du pair à travers la fonction `getAnnoucePresence` et dont le rôle est de parser et stocker les informations que contient ce pair. Une fois le tracker a validé cette étape, le pair a la possibilité de demander selon un ensemble de critères une liste de fichiers présents chez le tracker. La fonction `getLook` parse le premier argument qui est la requête du pair et cherche donc dans la liste de pairs stockés chez le tracker ceux qui vérifient les conditions imposées par le pair. Ce dernier peut aussi télécharger un certain fichier, c'est pour cela que nous avons implémenté la fonction `getFile` possédant le même prototype que la fonction getLook et dont le rôle est de fournir l'ensemble des pairs qui possédent ce fichier, le résultat est stocké dans le troisième argument.

4 Pair

L'implémentation des pairs a été fait en **Java**. Chaque pair doit pouvoir d'une part communiquer avec le tracker mais il doit également pouvoir communiquer et échanger des données avec les autres pairs.

4.1 Architecture de la partie Pair

La partie principale de l'architecture pour la partie Pair est représentée dans la Figure 2

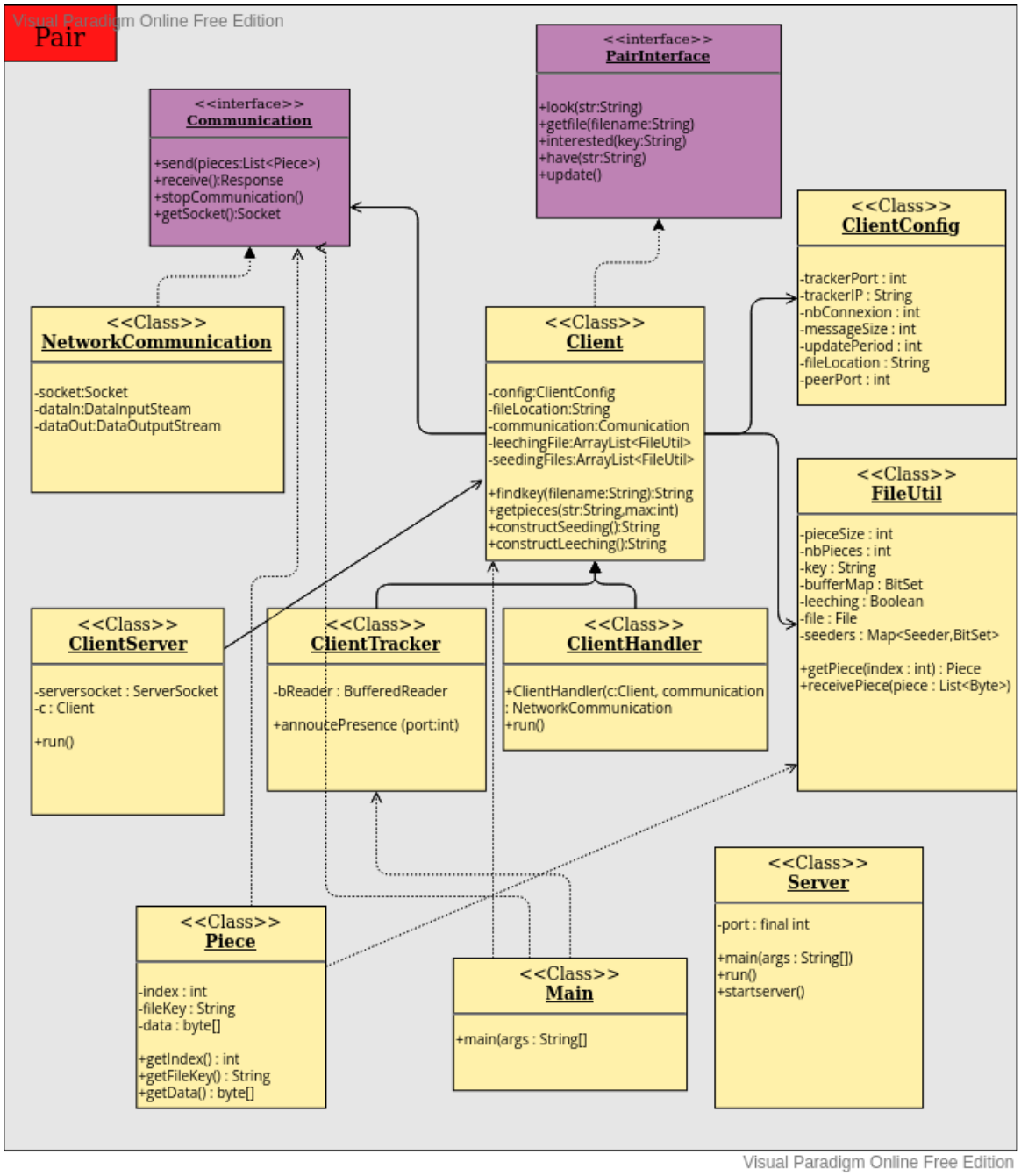

FIGURE 2 – Diagramme de classes

4.2 La configuration du pair

Afin de pouvoir faciliter le paramétrage d'un pair, un fichier texte *config.ini* a été mis en place. Les informations que nous retrouvons dans ce fichier sont les suivantes :

— l'adresse IP du tracker
— le port d'écoute du tracker
— le nombre de connexion maximum autorisé
— la limite sur la taille d'un message
— l'intervalle de temps au bout de laquelle nous faisons une mise à jour des données
— la localisation des fichiers à partager
— le port d'écoute du Pair

Chacune de ses informations est ainsi configurable selon le bon vouloir de l'utilisateur et elles vont être sauvegardées dans une instance de la classe *ClientConfig* qui permettra initialisation du Pair avec les configurations souhaitées.

4.3 Le stockage des fichiers

A l'instar du Tracker, chaque pair possède un ensemble de *FileUtil* où chaque *FileUtil* correspond à un fichier. Cet ensemble est décomposé en deux listes *leechingFiles* et *seedingFiles* avec les fichiers en cours de téléchargement et les fichiers possédés respectivement.

Les *FileUtil* fournissent diverses informations à propos du fichier comme son nom, sa taille, la taille des pièces, sa clé obtenue après hachage du fichier et son bufferMap. Le BufferMap permet la connaissance des parties de fichiers possédées par le Pair et correspond dans notre implémentation à une instance de *BitSet* qui est une classe de java.util. Il correspond à une séquence de bits où chaque bit correspond à une pièce. Si le bit est à 1, le pair possède la partie du fichier, sinon, il ne le possède pas.

4.4 Communication

La communication entre les entités du réseau est un aspect fondamental du projet. La classe *NetworkCommunication* implémentant l'interface *Communication* définit les fonction permettant d'envoyer et de recevoir des données à travers la socket stockée en attribut de cette classe. Ainsi chaque instance héritant de *Client* (un pair) va posséder un *NetworkCommunication* pour pouvoir communiquer.

4.4.1 Communication avec le Tracker

Afin de communiquer avec le Tracker, nous avons mis en place une classe *ClientTracker* qui hérite de la classe *Client* et dont la socket est associée au port et à l'adresse du tracker trouvés dans le fichier de configuration *config.ini*. Cette classe est instanciée dans la classe **Main** et lors de la première instanciation, nous demandons au pair de préciser les fichiers qu'ils possèdent parmi les fichiers situés dans le répertoire précisé dans le fichier de configuration. Une fois la connexion entre le pair et le tracker établie et la sélection des fichiers effectuée, ce pair va commencer par annoncer sa présence au tracker en stipulant son port d'écoute, la liste des fichiers qu'il possède et la liste des fichiers en cours de téléchargement. Cela est fait par le biais de la fonction **announcePresence()**.

```
< announce listen 34435 seed [file1.txt 207726 1024 5e34010cf5083240a21d0a2a986
5078c file2.txt 21 1024 ac0ee8c094236494f66db433f6c6d2ca] leech []
> ok
```

FIGURE 3 – requête getAnnoucePresence

Après cette étape d'annonce effectuée, le pair est libre de discuter avec le tracker. Il peut écrire en ligne de commande les diverses requêtes qu'il veut envoyer au tracker. On trouve plus particulièrement les commandes :

— **look [filename="$name"]** qui permet de chercher un fichier sur le réseau selon son nom.

```
< look [filename="file1.txt"]
> list [file1.txt 207726 1024 5e34010cf5083240a21d0a2a9865078c]
```

FIGURE 4 – requête look/list

— **getfile $key** qui permet de récupérer la liste de pairs (couple adresse ip, port) qui possèdent le fichier avec la clé correspondante.

```
< getfile a9937e4438b909ec438d89d7545d970e
> peers a9937e4438b909ec438d89d7545d970e [127.0.0.1:34009]
```

FIGURE 5 – requête getfile/peers

La communication avec le tracker est maintenu tout le long du dialogue.

4.4.2 Communication entre les pairs

En plus de la communication avec le tracker, les pairs présents dans le réseau doivent pouvoir communiquer et partager des fichiers entre eux. De ce fait, un pair doit alors pouvoir répondre aux requêtes venant d'autres pairs et doit ainsi écouter en permanence sur un port libre ou sur le port préciser dans son fichier de configuration. De plus, il doit pouvoir formuler des requêtes à d'autres pairs et donc pouvoir se connecter à d'autres pairs.

Connexion à un autre pair

Pour communiquer avec un autre pair, le pair va dans un premier temps établir une connexion avec le pair avec qui il veut communiquer. Il va ainsi entrer en ligne de commande la commande *connect* illustré en Figure 6

```
< connect 127.0.0.1:34009
> Succesfully connected
```

FIGURE 6 – requête de connexion

Une fois la connexion établie, divers commandes sont à sa disposition :

— **interested $key** qui permet au pair d'indiquer son intérêt pour un fichier.

— **getpieces key [$Index1 $Index2...]** qui permet de demander les différentes pièces d'un fichier selon leur disponibilité au niveau du pair.

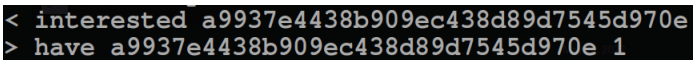

FIGURE 7 – requête interested/have

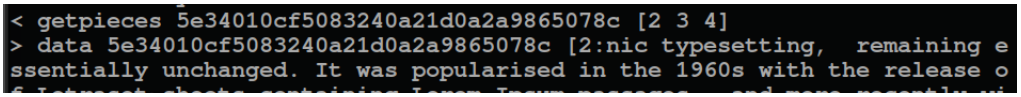

FIGURE 8 – requête getpieces/data

A noter que le pair peut continuer à interroger le tracker sans que cela ne perturbe son dialogue avec l'autre pair.

Écoute et réponses aux requêtes

Un pair doit se comporter comme le tracker, c'est à dire qu'il doit recevoir et répondre aux différentes requêtes venant d'autres pairs du réseau. Afin de satisfaire cette fonctionnalité, nous avons mis en place une classe *ClientServer* lancé en tant que thread et qui écoutera en permanence le port qui lui a été associé (qui est notamment le port annoncé au tracker) pour avoir connaissance de toute nouvelle connexion. Dès qu'une nouvelle connexion est accepté, le *ClientServer* va instancier un *ClientHandler* lancé en tant que thread et qui permettra la discussion avec le nouveau pair connecté. Il permet notamment de répondre aux requêtes évoquées dans la section 4.4.2 :

- **have $Key $BufferMap** en réponse à *interested* et qui envoie ainsi les partie du fichier qu'il possède (bufferMap).
- **data $Key [$Index1 :$Piece1 $Index2 :$Piece2 $Index3 :$Piece3 ...]** en réponse à *getpieces* et qui envoie les pièces demandées.

4.5 L'échange de fichiers

Le pair ayant formuler la requête *getpieces* à un autre pair reçoit alors en réponse un ensemble de pièces correspondant à des parties précise (désigné par leurs indexes) du fichier voulu. Le pair pourra alors reconstitué ces parties du fichier et ainsi être propriétaire lui aussi de ces parties de fichier, mettant donc à jour le BufferMap en rapport à ce fichier.

5 Divers outils

Nous avons mis en place divers outils afin de nous assurer du fonctionnement de notre implémentation et de nos fonctions.

5.1 Les tests

Des tests unitaires des fonctions élémentaires ont été implémentés afin de vérifier le fonctionnement de notre implémentation. Dans la partie du tracker, on vérifie le parsage et les réponses aux requêtes. Dans la partie du pair, on vérifie les instanciations des clients, le téléchargement avec l'envoi, la récupération de pièces d'un fichier et l'écriture dans un fichier.

5.2 Les logs

En plus des affichages sur le terminal représentant les échanges entre les différents entités, nous avons également mis en place des loggers qui met des informations supplémentaires sur

ces échanges dans un fichier *server.log* pour le tracker et *client.log* pour le pair.

6 Pistes d'amélioration

Dans cette partie, nous traitons les possibles pistes d'amélioration de notre projet.

6.1 Les fonctions périodiques

Pour commencer, une partie du sujet n'a pas été implémenté par manque de temps. Il s'agit des envois périodiques de la requête *update* au tracker et de la requête *have* aux pairs présent sur le réseau. Nous avons néanmoins implémenté ces fonctions.

6.2 Connexions simultanés des pairs

Ensuite, une amélioration possible serait de connecter le pair à tous les pairs possédant le fichier recherché et montrer son intérêt automatiquement. Pour le moment, nous devons nous connecter manuellement à un pair et faire cette demande nous-même.

6.3 La robustesse

Dans l'état actuel de notre projet, notre application manque de robustesse. En effet, il existe de nombreux cas où l'application plante car nous n'avons pas eu le temps de traiter tous les scénarios possibles.