**Projet BitTorrent**

**Auteurs : Da Silva Andrade David**

**Magnin Antoine**

**Classe : ITI-Jour 3ème année 2014-2015**

Table des matières

[2 L’objectif du projet 2](#_Toc408164289)

[3.1 La terminologie 2](#_Toc408164290)

[3.2 L’encodage 2](#_Toc408164291)

[3.3 Les pièces et les blocks 3](#_Toc408164292)

[3.4 Le fichier de métadonnées 4](#_Toc408164293)

[3.5 Le tracker HTTP 4](#_Toc408164294)

[3.6 Le protocole « Peer Wire » 4](#_Toc408164295)

[4 Approche pratique de l’implémentation du client BitTorrent 5](#_Toc408164296)

[4.1 Vue globale 5](#_Toc408164297)

[4.2 Le traitement du fichier de métadonnées 5](#_Toc408164298)

[4.3 Allocation de l’espace disque et requête au Tracker 6](#_Toc408164299)

[5 Sources 7](#_Toc408164300)

# L’objectif du projet

Ce projet a pour but l’implémentation d’un client BitTorrent simplifié, il doit simplement se connecter à un tracker dont l’adresse est fournie dans un fichier .torrent. Une fois l’adresse de ce tracker extraite il faut effectuer une requête afin d’obtenir les ips et les ports des peers qui ont les pièces du torrent. Une fois ces ip stockées il faut démarré la communication avec les peers, pour ce faire on utilise le protocole PWP, celui-ci permettra de savoir qu’elles pièces contiennent chaques peers, ainsi que la demande de celle-ci. Le client doit pouvoir garder un trace de chaque pièce que possèdent les peers, ainsi qu’effectué des logs de ceux-ci dans un forma ASCII et CSV.

Il n'est pas obligatoire que votre implémentation soit capable de récupérer la totalité d'un fichier partagé. Elle devra cependant déterminer sur quels pairs se trouvent quels morceaux de fichiers, récupérer les deux premiers morceaux (pieces) du fichier et vérifier que la fonction de hachage SHA-1 sur les morceaux récupérés correspond bien aux résultats indiqués dans le fichier des métadonnées .torrent.

Notre client est un *bad leecher*, c’est-à-dire qu’il ne partagera pas les pièces qu’il a en sa possession.

Approche théorique du protocole

## La terminologie

**Peer** : un pair est un nœud dans le réseau qui participe au partage de fichier. Il peut agir simultanément en tant que serveur ou en tant que client avec les autres nœuds du réseau.

**Neighboring peers** : Pairs à qui un client a un point actif à point TCP.

**Client** : Un client est un agent utilisateur qui agit comme un pair pour le compte d'un utilisateur.

**Torrent** : Un torrent est le terme pour le fichier (un seul fichier torrent) ou un groupe de fichiers (multi-fichiers torrent) que le client télécharge.

**Swarm** : Un réseau de pairs qui opèrent activement sur un torrent donné.

**Seeder** : Un pair qui a une copie complète d'un torrent.

**Tracker** : Un tracker est un serveur centralisé qui contient des informations sur un ou plusieurs torrents et des réseaux de pairs associés. Il fonctionne comme une passerelle pour les pairs dans un réseau de pairs.

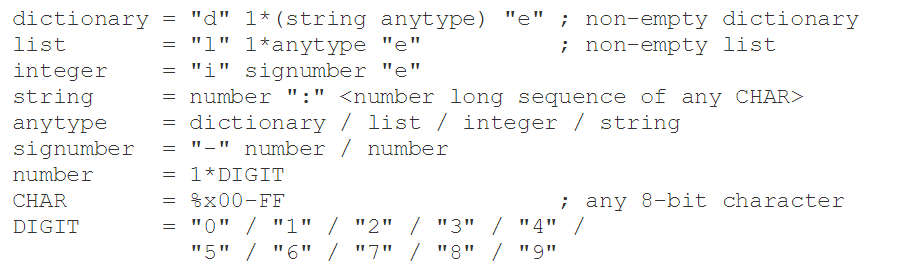
**Metainfo file** : Un fichier texte qui contient des informations sur le torrent, par exemple, l'URL du tracker. Il a généralement le .torrent d'extension.

**Peer ID**: Une chaîne de 20 octets qui identifie le poste. Comment l'ID des pairs est obtenu en dehors de la portée de ce document, mais un poste doit faire en sorte que l'ID de pairs qu'il utilise a une très forte probabilité d'être unique dans le réseau de pairs.

**Info hash**: Un hash SHA1 qui identifie de manière unique le torrent. Elle est calculée à partir de données dans le fichier de métadonnées.

## L’encodage

Dans BTP / 1.0 le fichier métadonnées et toutes les réponses du Tracker sont encodés dans le format de bencoding. Le format spécifie deux types scalaires (entiers et chaînes) et deux types de composés (listes et dictionnaires).



Les entiers sont codés en préfixant une chaîne contenant la représentation en base dix du nombre entier avec la lettre "i" et postfixant avec la lettre "e". Par exemple l'entier 123 est codé comme i123e.

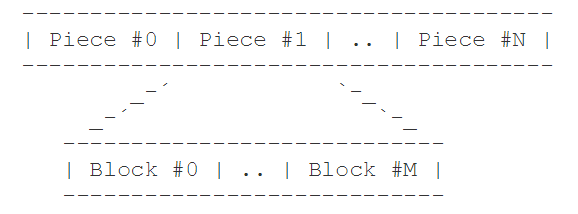
Les chaînes sont encodées en faisant précéder le contenu de la chaîne avec la longueur de la chaîne suivi de deux points. Par exemple la chaîne "announce" est codé comme "8: announce".

Les listes sont un nombre arbitraire d'éléments qui sont « bencodé » précédés de la lettre "l" et postfixé avec la lettre "e". Il en résulte que les listes peuvent contenir des listes imbriquées et des dictionnaires. Par exemple "li2e3: foo" définit une liste contenant le nombre entier "2" et la chaîne "foo".

Les dictionnaires sont un nombre arbitraire de paires clé / valeur délimités par la lettre "d" au début et à la lettre "e" à la fin. Toutes les touches sont bencoded cordes tandis que la valeur associée peut être tout élément bencoded. Par exemple " d5:monthi4e4:name5:aprile" définit un dictionnaire contenant les associations: "month" => "4"et "name" => "april". Toutes les clés de dictionnaire doivent être triés.

## Les pièces et les blocks

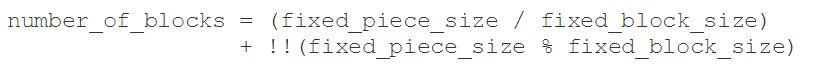
Chaque pièce représente une série de données dont il est possible de vérifier en utilisant un hachage SHA1 pièce. Lors de la distribution des données sur pièces PWP (Peer Wire Protocol) sont divisés en un ou plusieurs blocs, comme le montre le schéma ci-dessous:



Le nombre de pièces dans le torrent est indiqué dans le fichier de métadonnées. La taille de chaque pièce dans le torrent reste fixe et peut être calculée selon la formule suivante:



La taille d'un bloc est définie une valeur choisie lors de l’implémentation qui ne dépend pas de la taille fixe de la pièce. Une fois une taille fixe définie, le nombre de blocs par pièce peut être calculé en utilisant la formule:



## Le fichier de métadonnées

Le fichier métadonnées fournit au client des informations sur la localisation du tracker ainsi que le torrent à télécharger. Outre la liste des fichiers qui se traduira par le téléchargement de torrent, il répertorie également comment le client doit séparer et vérifier les pièces individuelles qui composent le torrent complet.

## Le tracker HTTP

Le protocole HTTP Tracker (THP) est un mécanisme simple pour introduire des pairs entre eux. Un tracker est un service HTTP qui doit être contacté par un pair pour rejoindre un réseau de pairs. En tant que tel le tracker constitue le seul élément centralisé en BTP / 1.0. Un tracker ne fournit pas par lui-même l'accès à toutes les données téléchargeables. Un tracker de pairs repose sur l'envoi des demandes régulières. Il peut supposer qu'un pair est mort s’il manque une requête.

## Le protocole « Peer Wire »

Le but du PWP, est de faciliter la communication entre pairs voisins dans le but de partage de fichiers. PWP décrit étapes prises par un pair après qu'il a lu dans un fichier métadonnées et contacté un tracker pour recueillir des informations sur d'autres pairs avec lesquels il peut communiquer. PWP est posée sur le dessus du protocol TCP et gère toute sa communication en utilisant des messages asynchrones.

# Approche pratique de l’implémentation du client BitTorrent

## Vue globale

Pour la mise en place de notre client nous avons opté pour l’utilisation du Java comme langage de dévelloppement, il apport une plus simple implémentation des sockets que le C, ainsi qu’une plus grande facilité pour la gestion des chaînes de caractères.

## Le traitement du fichier de métadonnées

Comme vue dans l’approche théorique, le fichier contenant les métadonnées est encodé en Bencode. Afin de décodé sont contenu nous avons utilisé une librairie qui se nomme bee-encode :

<https://code.google.com/p/bee-encode/>

Cette librairie nous permet de simplement d’éxtraire les informations du fichier « .torrent » en passant le flux de donnée (*InputStream*) éxtrait de l’ouverture du fichier « .torrent ». Les dictionnaires sont vus comme un type Map sous Java, les entier sont des BigInteger et les listes sont des ArrayList.

Pour extraire toutes les informations utiles du fichier nous avons créer un classe « Metafile », celle-ci permet simplement en lui passant un fichier au constructeur d’extraire toutes les données et de les rendres ensuites accèssibles via des accesseurs. Voici la liste des paramètres accéssibles ainsi que leurs types :

* String announce
* ArrayList<String> announce\_list
* String comment
* String createdBy
* Date creationDate
* Map<String, ?> info
* private String name
* Integer piece\_length
* byte[] pieces
* String source
* Integer length (si c’est un seul fichier)
* ArrayList<Map<String, ?>> files (si c’est un torrent de type multifile)

Les fichiers torrents qui contiennent plusieurs fichiers (multifiles) peuvent être tester via une méthode de la classe Metafile qui se connect « isSingleFile() », qui retourne un boolean à faux si c’est le cas.

## Allocation de l’espace disque et requête au Tracker

Une fois les informations du contenu du torrent extraites des métadonnées via la classe Metafile, il faut alloué l’espace disque où l’on mettra ensuite toutes les pièces téléchargées avec notre client. Pour cela nous utilisons une classe qui se nomm « Torrent », celle-ci prend 3 paramètres pour le constructeur. Le premier et l’objet de type Metafile précédement créer. Le deuxième est le numéro du port avec lequel on désire communiquer avec le tracker afin d’effecter la requête des peers. Et finalement, le troisième, c’est le peer ID de notre client, celui-ci peut être choisi aléatoirement, il faut simplement qu’il soit composé de 20 chiffres.

Si le fichier s’agit d’un type « single file » on va simplement créer un seul fichier comportant le nom du torrent. Si c’est un torrent dit « multifiles », un dossier comportant le nom du torrent sera créer, il contiendra ensuite toutes l’arborescense du torrent.

L’infohash est également calculé dans le constructeur de cette classe, il sera nécessaire pour effectuer la requête au tracker. Il est accessible à tout moment via un accesseur « getInfoHash() » qui retourne celle-ci sous forme d’un tableau de bytes.

Pour la requête vers le tracker on utilise la méthode « request » de la classe Torrent, celle-ci va effectuer en envoyant une simple URL contenant toutes les informations nécessaires, en voici sa construction :

URL url = **new** URL(torrent.getAnnounce() +

"?info\_hash=" + **this**.infoHashEncoded +

"&peer\_id=" + **this**.peerID +

"&port=" + **this**.port +

"&uploaded=" + **this**.uploaded +

"&downloaded=" + **this**.downloaded +

"&left=" + **this**.left +

"&event=" + **this**.event +

"&key=12345" + "&compact=1");

L’infohash est celui qui a été calculé précédemment. Le peer ID c’est celui qui à été passé lors de la construction de l’objet, idem pour le port. Les champs uploaded et downloaded sont à 0 car on part du principe que c’est la première fois qu’on télécharge le fichier (donc pas de pièces présentes sur le disque) et aussi que notre client est un bad leecher, donc qu’il n’a pas uploadé de pièces. Le champ left correspond à quantité totale de bytes que le peer (notre client dans ce cas) nécessite pour finalisé sont téléchargement. Le champ event dans notre cas est utilisé avec la valeur « started », celle-ci doit toujours être mise si c’est la première fois qu’un peer se connecte au tracker.

Les deux derniers champs, key et compact, ne sont pas toujours nécessaires, mais afin de faire en sorte que notre requête fonctionne vers le plus grand nombre de trackers possibles nous avons décidé de les garder. Le paramètre key est une clé qui est censé prouvé l’authenticité de l’auteur de la requête, dans notre cas nous avons juste mis une valeur fixe. Le compact est pour décrire sous quel format nous demandons la réponse qui contient les peers. Si le compact vaut 0, la réponse sera sous la forme d’un fictionaire contenant l’ip des peers, leur port ainsi que l’ID de ceux-ci. Si le compact est à 1 la réponse sera simplifiée, les peers seront « encodé » sous la forme de 4 bytes pour l’adresse ip, suivi de 2 bytes pour le port. Chaque peer sera donc espacé de 6 bytes. L’avantage de l’utilisation du compact c’est que la trame de réponse se retrouve être considérablement raccourcie car l’ID du peer n’est pas présent. Si on désire vraiment le connaître il faudra l’extraire lors du handshake avec le peer en question.

La méthode request retourne un dictionnaire contenant la réponse complete

## La liste des peers

## Le gestionnaire de peers

# Sources

Pour toutes les définitions théoriques du protocol :

<http://www.bittorrent.org/beps/bep_0003.html>

<https://wiki.theory.org/BitTorrentSpecification>

<http://jonas.nitro.dk/bittorrent/bittorrent-rfc.html>