Projet Réseau 3A SEOC

Client bittorrent

SEOC 3A

0 Projet non rattrapable et Fraude

Il est interdit d'utiliser le projet d'une autre équipe, ou de mettre à disposition son projet à une autre équipe. Vous êtes encouragés à aller chercher de l'information sur internet, mais pas à copier/coller du code que vous n'avez pas écrit vous même. La sanction en cas de fraude va d'au minimum la note 0/20 au projet jusqu'à une sanction disciplinaire (plus de détails sur Règlement INP. Pour plus de détails, voir la charte des projets de l'Ensimag, qui s'applique aussi à ce projet : http ://intranet.ensimag.fr/teide/Charte_contre_la_fraude.php.

En cas de doute, n'hésitez pas à demander conseil.

Le seul code que vous pouvez ré-utiliser sera pour du code pour traiter l'encodage Bencode utilisé dans les fichiers Torrent.

1 Organisation projet

Le projet consiste en la programmation complète d'un protocole réseau à travers le développement d'un client bittorrent en Java capable d'interagir avec un tracker existant et des clients bittorrent existant capable de télécharger/uploader un fichier (une distribution linux par exemple) décrit par un fichier torrent.

Le projet est à réaliser en équipe de 3 étudiants.

Le projet est à réaliser en dehors et durant les séances réservées dans ADE (12 + l'évaluation finale). Cela représente environ 120h [1] de travail personnel par étudiant (séances incluses).

Les séances encadrées seront dédiées à vos questions, à quelques présentations de concept et à des points d'avancement.

1.1 L'objectif : le client bittorrent

BitTorrent est un protocole de partage de fichier pair-à-pair. Dans ce projet, nous suivrons la spécification détaillée à http ://www.bittorrent.org/beps/bep_0003.html afin de développer un client capable de lire un fichier torrent (pas de multi-fichier), de dialoguer avec un tracker existant et interagir avec des clients pour échanger les pièces du fichier à télécharger/uploader. Plus de ressources en Section 2.3

Vous lancerez votre client avec la syntaxe suivante : `java -jar mybittorrent.jar [-debug] [-info] <file.torrent> <download folder>` :

- `fichier.torrent` chemin vers le fichier torrent
- `dossier de téléchargement` le répertoire où se trouve le fichier à échanger
- `-debug` (optional) les informations minimales de debug
- `-info` (optional) affiche toutes les secondes les informations vis-à-vis des pairs (bittorrent application, adresse IP, port, état de download/upload des différentes pieces)

Le fichier décrit par le torrent est téléchargé et uploadé jusqu'à ce que le processus soit arrêté.

Vous pourrez ré-utiliser une bibliothèque pour l'encodage/décodage Bencode (bibliothèque Bencode par exemple).

1.2 Concepts avancées de programmation, bonnes pratiques de développement

Ce projet est l'occasion tout simplement de gagner en expérience en

- consolidant vos acquis (POO, Réseau 2A) en les mettant en application sur un projet de grande envergure faiblement cadré et sans code de départ (Architecture Orienté Objet)
- abordant des concepts de programmations avancées (Réseau, Design Pattern)

et de développer des bonnes pratiques de développement en terme de

1. 25-30h x 4 ECTS

— gestion d'équipe/de projet (Méthode Scrum)
— un code propre (Clean Code, Factorisation, Architecture Objet)
— une utilisation avancé de git (git workflow, des commit propres, ...) avec gitlab

Le propre de tous ces concepts est que, sans les mettre en pratique en conditions réelles, il est difficile de se faire une idée. C est l'opportunité de vous tester !

1.3 Déroulement indicatif du projet

Sem.	Séance	Contenu
39	0 : Choix du projet	Présentation Projet/Scrum + mise en place gitlab
40	1 : le protocole	Etude du protocole bittorrent (biblio + plateforme)
41	2 : Fin Sprint 0 (20h) (cf Section 1.4)	Description d'un échange bittorrent + démarrage Sprint 1
42	3	
43	4	
44	VACANCES	
45	5	
46	6 : Fin Sprint 1 (+140h) (cf Section 1.5)	Client - tracker
		Leecher 0% (client) - Seeder (Vuze)
		Seeder 100% (client) - Leecher (Vuze)
47	SEM. PROJET	
48	7 : TP "Clean Code"	Intervenants extérieurs
49	8	
50	9	
51	10 : Fin Sprint 2 (+120h) (cf Section 1.6)	Multi-client : client - plusieurs Vuzes
		Stratégie de téléchargement
		Resume
3	11	
4	12 : Fin Sprint 3 (+80h) (cf Section 1.7)	Peaufiner le client
		...
21/1/21	Rendu final	Code + Rapport (Section 1.9)
5	Soutenance finale	Démo + Questions

Ce déroulé n'est qu'indicatif. Vos sprints évolueront en fonction de votre avancement. Vous serez certainement amenés à revoir les User Stories.

Vous devrez présenter votre avancement (revue de sprint) avec rendu éventuel aux dates de fin de sprint indiquées :

— Sem. 42 : revue de sprint 0 (cf. section 1.4.1)
— Sem. 47 : revue de sprint 1 (cf. section 1.5.4)
— Sem. 51 : revue de sprint 2 (cf. section 1.6.5)
— 23/01/2020 : Rendu de code + Rapport
— Sem. 5 : Soutenance (Démonstration + Questions)

1.4 Sprint 0 : Etude du protocole bittorrent (1 pt)

Ce qui est attendu pour le sprint 0 :

— L'etude du protocole bittorrent (cf. section 2.3.1)
— Mise en oeuvre d'une plateforme de test en salle D200/D201 avec par exemple :
  — Créer votre torrent sous Vuze par exemple
  — sur une machine
    — un tracker bittorrent (par exemple opentracker) [2]
    — et un client bittorrent (Vuze par exemple ou transmission sous OSX)) qui fera office de Seeder (détient le fichier à 100%)
  — et, sur la même machine ou une autre machine, un client bittorrent qui fera office de leecher (détient le fichier à 0%)
— une capture wireshark d'un échange bittorrent d'une session basique obtenue sur cette plateforme

2. Le parefeu de l'Ensimag peut vous empêcher d'interagir avec des trackers extérieurs (à vérifier)

1.4.1 Revue de sprint

La revue de sprint consistera en une présentation de 10 min de votre compréhension du protocole bittorrent avec :

- un diagramme temporel décrivant les différentes phases, l'enchainement des messages de la capture wireshark et leur signification
- une explication du contenu des messages les plus importants \footnote{vous avez la possibilité d'exporter au format texte votre capture dans wireshark via `Fichier > exporter analyse des paquets - as plain text`
- une rétrospective sur le sprint 0 (état du tableau des tâches, burn down chart)
- et pour les plus avancés (facultatif) : ce que vous avez prévu pour le sprint 1 (découpage en tâches et estimation)

1.5 Sprint 1

Vous ne devez supporter que les torrents mono-fichier (**pas de torrent multi-fichier**).

Afin de faciliter le développement en local sur votre machine, vous allez devoir faire tourner l'équivalent de la plateforme distribué en local sur votre machine en faisant dialoguer Vuze, Opentracker et votre application à travers l'interface virtuelle de loopback `lo0` de votre système d'exploitation et en capturant le trafic sur cette dernière.

1.5.1 leecher 0% (3pts)

Cette étape consiste en le développement de la partie de votre client qui télécharge les pièces auprès d'**un client** existant (le cas de plusieurs clients sera traité au sprint 2). Cette étape est remplie dès que votre client démarre sans aucune pièce du fichier ("leecher 0%") et télécharge 100% des pièces auprès d'un client Vuze (par ex.) qui jouerait donc le rôle de "seeder 100%". Le fichier doit se composer de plusieurs pièces.

Cette étape nécessitera :

- l'utilisation de sockets bloquantes
- la construction et l'interprétation des différents messages

Pour des exemples, se reporter au chapitre 3 du livre *"TCP/IP sockets in Java : practical guide for programmers"*

Si vous souhaitez passer directement aux sockets non bloquantes et leurs *Buffers* associés (notions plus avancées), aller voir la section 1.6.1 du sprint 2.

Cette étape peut être codée sans échange préalable avec le tracker.

1.5.2 dialogue avec le tracker (1.5pts)

Dans cette étape, vous devez être en mesure d'émettre une requête HTTP à partir du fichier torrent vers le tracker et récupérer les adresses IP et port des pairs souhaitant participer aux échanges de pièces de ce fichier.

Vous pourrez ré-utiliser une bibliothèque pour l'encodage/décodage Bencode (bibliothèque Bencode par exemple).

1.5.3 seeder 100% (2pts)

Vuze agit cette fois-çi en tant que "leecher 0%" et télécharge 100% des pièces auprès de votre client.

1.5.4 Revue de sprint

La revue de sprint consistera en une présentation de 10 min de ce qui a été réalisé durant le sprint 1 ou de l'avancement du sprint 1 (avancement + explications des fonctionnalités techniques importantes + rétrospective du déroulement du sprint).

1.6 Sprint 2

1.6.1 Multi-clients : Votre client et plusieurs Vuze (3pts)

Le scénario qui peut servir de guide pour cette étape est le suivant : votre client démarre sans aucune pièce et doit se connecter et récupérer simultanément plusieurs pièces de plusieurs clients existants.

Pour ce faire, vous devrez utiliser les sockets non bloquantes (java.nio) qui est bien plus efficace que les solutions multi-threadés à base de sockets bloquantes et les *Buffer* de ces sockets.

Pour des exemples, se reporter au chapitre 5 du livre *"TCP/IP sockets in Java : practical guide for programmers"*

1.6.2 Algorithme de sélection des pièces (1pts)

L'algorithme devra respecter les contraintes suivantes :

- ne pas envoyer de requête pour une pièce qu'un peer ne possède pas
- ne pas envoyer des requêtes pour le même bloc à des peers différents
- ne pas requérir une pièce que vous avez déjà

1.6.3 Resume : interruption et reprise de téléchargement (1pts)

En redémarrant, votre client doit être en mesure de reprendre le téléchargement là où il a été interrompu.

1.6.4 Evaluation de performance (1pts)

Evaluer les performances (vitesse de téléchargement principalement, mémoire, CPU) de votre application lors du téléchargement d'une distribution linux par exemple à plusieurs.

1.6.5 Revue de sprint}

La revue de sprint consistera en une présentation de 10 min de ce qui a été réalisé durant le sprint 1 ou de l'avancement du sprint 2 (avancement + explications des fonctionnalités techniques importantes + rétrospective du déroulement du sprint).

1.7 Sprint 3 : peaufiner votre code

Le coeur du projet est d'arriver au bout du sprint 2 avec une architecture orientée objet qui tient la route et un code propre et pertinemment commenté. Si ce coeur est réalisé, vous pouvez envisager d'implémenter les extensions proposées ci-dessous.

1.7.1 Algorithmes de sélection des pièces avancés (1pt)

Cette étape consistera en la mise en place d'algorithme de sélection de piece plus avancé que séquentiel ou aléatoire comme "Rarest First", "Endgame mode" ou le choix de pairs selon d'autes critères (RTT par ex.).

1.7.2 Implémenter 2-3 design pattern (1.5pt)

Cette étape consistera en l'implémentation de patron de conception au sein de votre code. Ainsi on pourra s'intéresser au "Réacteur", "Observateur" ou la mise en place d'une machine à état ou d'autres patrons qui vous semble pertinent pour notre client.

1.7.3 Clean Code (1.5pt)

Mettre en oeuvre quelques principes du clean code vu en TP ou le livre Coder proprement que vous pourrez trouver en plusieurs exemplaires (anglais , français) à la bibliothèque universitaire Joseph Fourier.

1.8 Bonus

Pour ceux qui veulent aller plus loin

1.8.1 Evaluation automatisée et reproductible de performance

Cette étape consistera en la mise en place de tests distribués. Afin de mettre en évidence la supériorité de votre stratégie de téléchargement, il va falloir mettre en place des scénarios impliquant plusieurs clients distribuées sur plusieurs machines de la salle D200/D201. Afin de pouvoir comparer une expérience à une autre, il faudrait pouvoir automatiser la mise en oeuvre de ses scénarios en pilotant de lancement de différents Vuze sur différents machines qui échangeraient avec votre client. En comparant les temps de téléchargement obtenus dans la même situation, vous pourriez mettre en évidence l'efficacité de vos stratégies de téléchargement les unes par rapport aux autres. Et pourquoi pas comparer l'efficacité de plusieurs clients de différents groupes ?

1.8.2 1 extension de protocole parmi plusieurs

Implémenter une extension parmi les suivantes :

- Optimistic unchoking : à la fin de \url{https ://www.bittorrent.org/beps/bep_0003.html
- Peer Exchange (PEX) : http://www.bittorrent.org/beps/bep_0011.html

— Local Service Discovery : http://www.bittorrent.org/beps/bep_0014.html
— Micro Transport Protocol (µTP) : \url{http ://bittorrent.org/beps/bep_0029.html
— The Azureus messaging protocol : https://wiki.vuze.com/w/Azureus_messaging_protocol
— DHT (Bittorrent sans tracker centralisé) : http://www.bittorrent.org/beps/bep_0005.html

Attention les extensions ne sont certainement pas équivalentes en terme de difficulté d'implémentation. Prenez celle qui vous semble la plus intéressante (i.e. vous apprenant quelque chose de nouveau).

1.9 Rendu de code + rapport

Votre code devra s'accompagner dans le dépôt git d'un README.md indiquant

1. Comment obtenir le `mybittorrent.jar` utilisé ci-dessous à partir de votre git (librairies à inclure, . . .)
2. comment lancez votre application (je rappelle la syntaxe de lancement des consignes : java -jar mybittorrent.jar [–debug] [—info] file.torrent download_folder
   — vous fournirez un fichier torrent et le fichier source pour tester ou le logiciel que vous avez utilisé pour créer le fichier torrent
3. l'application existante avec laquelle vous avez tester votre application (Aucune, la votre, vuze, . . .)
4. le tracker utilisé (en local ou distant)
5. le scénario de test (leecher/seeder, en local/sur plusieurs ordinateurs, quel port d'écoute, . . .) que vous me conseillez.

Le rapport devra suivre les consignes suivantes

— rendu du rapport sur votre git dans le répertoire « doc ».

2 Architecture hints

A BitTorrent client is any program that implements the BitTorrent protocol. Each client is capable of preparing, requesting, and transmitting any type of computer file over a network, using the protocol. A peer is any computer running an instance of a client.

To share a file or group of files, a peer first creates a small file called a "torrent" (e.g. MyFile.torrent). This file contains metadata about the files to be shared and about the *tracker*, the computer that coordinates the file distribution.

Peers that want to download a file must first obtain a torrent file for it, and connect to the specified tracker, which tells them from which other peers to download the pieces of the file. The client connects to those peers to obtain the various pieces. After the file is successfully and completely downloaded, the peer is able to shift roles and become an additional seed, helping the remaining peers to receive the entire file.

2.1 Bencoding

Bencoding is a way to specify and organize data in a terse format. It supports the following types :

Type	Encoding	Example
byte strings	string length encoded in base ten ASCII :string data	4 :spam
integers	integer encoded in base ten ASCII	i3e
lists	bencoded values	l4 :spam4 :eggse
dictionaries	bencoded string bencoded element	d3 :cow3 :moo4 :spam4 :eggse

2.2 Metainfo File Structure

All data in a metainfo file is bencoded. The content of a metainfo file (the file ending in ".torrent") is a bencoded dictionary, containing the mandatory keys listed below. All character string values are UTF-8 encoded.

— **info** : a dictionary that describes the file(s) of the torrent. There are two possible forms : one for the case of a 'single-file' torrent with no directory structure, and one for the case of a 'multi-file' torrent
— **announce** : The announce URL of the tracker (string)

For the case of the *single-file mode*, the **info** dictionary contains the following mandatory fields :

— **name** : the filename. This is purely advisory. (string)

— **length** : length of the file in bytes (integer)
— **piece length** : number of bytes in each piece (integer)
— **pieces** : string consisting of the concatenation of all 20-byte SHA1 hash values, one per piece (byte string, i.e. not urlencoded)

2.3 Resources (non exhaustives)

2.3.1 Bittorrent

— The BitTorrent Protocol Specification http ://www.bittorrent.org/beps/bep_0003.html
— Unofficial Bittorrent Protocol Specification v1.0 https ://wiki.theory.org/index.php/BitTorrentSpecification
— BitTorrent Protocol http ://en.wikipedia.org/wiki/BitTorrent_%28protocol%29
— BitTorrent vocabulary http ://en.wikipedia.org/wiki/Terminology_of_BitTorrent