TP3 : projet pair à pair

Ce TP durera (au moins) trois semaines, soit trois fois 2h40. Il vous demandera probablement un peu de travail en dehors des heures de TP, mais en étant efficaces cette part du travail devrait être relativement réduite. L'objectif sera d'implanter le protocole pair à pair *Chord* vu en cours.

Quelques points importants. Ce TP sera noté, il représentera *au moins un tier de votre note finale* du module. Vous devrez me rendre *un rapport* présentant et justifiant vos choix d'implantation et me faire la démonstration du fonctionnement de votre réseau en *le déployant sur l'ensemble des machines d'une salle de TP*. Vous me rendrez aussi *votre code* pour que je puisse vérifier qu'il correspond bien à ce qui est expliqué dans votre rapport, cependant la qualité du code ne rentrera pas ou peu en compte dans votre note (ceci *ne vous dispense pas de documenter ce code*).

Pour le langage à utiliser je recommande Java, cependant le choix est libre. Veillez quand même à choisir un langage avec lequel vous êtes à l'aise et qui fournisse des bibliothèques pour les sockets et les threads. Je vous recommande aussi d'éviter C, avec lequel la manipulation des sockets peut s'avérer délicate. Vous devez garder en tête que vous n'avez pas affaire à un exercice de programmation, mais plutôt à un exercice de prototypage : l'objectif est de mettre en œuvre un protocole, dans le but de pouvoir mieux apréhender son comportement et d'analyser ses performances.

La suite de ce document décrit, plus en détails, le travail que vous aurez à faire. Vous devez réaliser au moins l'une des extensions proposées ou bien une autre extension de votre choix (que vous me soumettrez avant de commencer sa programmation, pour que je la valide).

1 Ce qui vous est fourni

Pour vous faciliter le travail, je vous fourni un certain nombre d'outils, sous forme de programmes Java, qui sont décrits dans cette partie. Ils peuvent contenir des bugs, merci de me signaler tout problème pour que je puisse corriger. Ces outils sont un peu rudimentaires, si vous voyez des fonctionnalités utiles qui n'y sont pas intégrées, signalez le moi.

Serveur d'accueil. Comme vu en cours, lorsqu'un pair souhaite être intégré à un réseau implantant le protocole *Chord*, il a besoin de pouvoir contacter un pair déjà présent dans le réseau. Cette découverte d'un pair peut s'avérer complexe en l'absence de gestion centralisée du réseau. Le serveur d'accueil implanté par *WelcomeServer.java* vous simplifie cette tâche. Vous avez juste à démarrer ce serveur en lui indiquant quel port utiliser pour le contacter et la taille maximale du réseau logique que vous utiliserez : `java WelcomeServer port taille`. Ensuite, quand un pair veut entrer dans le réseau il contacte le serveur d'accueil en lui envoyant son ip et le hachage de celle ci par un message de la forme suivant : `yo:hash:ip`. Attention, il faut absolument que le serveur reste allumé pendant toute la durée de vie de votre réseau et il faut démarrer un nouveau serveur à chaque fois que vous voulez créer un nouveau réseau.

Serveur de hachage. Les fonctions de hachage fournies par Java peuvent s'avérer complexes à utiliser et fournissent des valeurs dans des espaces très grands (trop grands pour faire des tests de petits réseaux implantant le protocole *Chord*). Le serveur de hachage implanté par *HashServer.java* vous fournira des hachages. Vous avez juste à démarrer ce serveur en lui indiquant quel port utiliser pour le contacter et la taille maximale du réseau logique que vous utiliserez : `java HashServer port taille`. Ensuite, quand un pair veut obtenir le hachage d'une adresse ip il lui suffit d'envoyer cette ip au serveur de hachage.

Attention, il faut absolument que le serveur reste allumé pendant toute la durée de vie de votre réseau. Par contre, il est inutile de le redémarrer à chaque nouveau réseau.

Moniteur. Il peut s'avérer compliqué d'observer les différents pairs de votre réseau. Le moniteur implanté par *MonitorServer.java* vous aidera dans cette tâche. Vous avez juste à démarrer ce serveur en lui indiquant quel port utiliser pour le contacter ainsi que les informations permettant de contacter le serveur d'accueil : `java MonitorServer welcomeHost welcomePort port`. Vous pouvez ensuite interagir avec le moniteur dans le terminal. Tapez simplement `help` pour voir la liste des commandes disponibles. Le moniteur peut être démarré ou éteint n'importe quand.

2 Votre travail

Votre objectif principal sera de programmer une version de *Chord*, tel qu'il a été décrit en cours. Pour plus d'informations vous pouvez aller consulter l'article de recherche associé, qui est disponible sur Madoc. Vous ne vous préoccuperez pas (du moins dans un premier temps) de la gestion des données ni du départ éventuel de pairs. Il vous faudra donc proposer une implantation d'un pair capable de :

- transmettre (efficacement) un message à un autre pair à partir de l'identifiant de celui-ci,
- prendre en compte l'arrivée d'un nouveau pair dans le réseau.

Pour cela il vous faudra en particulier définir une politique de mise à jour des tables de routage des pairs. Vous ferez particulièrement attention à décrire et justifier soigneusement dans votre rapport la politique que vous aurez choisie.

Contraintes imposées par les outils fournis. Pour que les outils que je vous fourni puissent fonctionner, il faut respecter un certain nombre de contraintes.

- Votre implantation d'un pair doit assurer que, à tout moment, chaque pair écoute sur le port du moniteur. En cas de connexion du moniteur il faudra répondre à sa requête, c'est-à-dire lui envoyer la table de routage maintenue par votre pair, en respectant le protocole de fonctionnement du moniteur. Il s'agira d'envoyer les lignes de votre table une à une, selon la forme `k:succ:ip` où k est la position sur le réseau logique à laquelle correspond cette ligne de la table, succ est le premier pair dans le réseau qui se trouve après k et ip est l'ip de ce pair. Après la dernière ligne il faudra envoyer le message `end`.
- Tout pair qui entre dans le réseau doit se signaler au serveur d'accueil.

Pour plus de détails, n'hésitez pas à regarder les codes fournis.

3 Extensions

Parmi les extensions envisageables on peut citer :

- Gérer le départ, attendu ou non, d'un pair.
- Gérer des données sur le réseau.
- Permettre l'analyse des performances de votre réseau.
- Implanter un algorithme de consensus réparti entre les pairs de votre réseau (on considère qu'il n'y aura pas de pannes)

Vous devez implanter au moins une extension. Faites bien attention à prendre en compte les outils que je vous fourni lors de ce travail : plusieurs extensions impliquent de modifier ceux-ci (par exemple les données doivent probablement être gérées de façon différente des pairs par le serveur d'accueil pour permettre au moniteur de continuer à fonctionner correctement) ou de prendre en compte certaines de leur fonctionnalités non utilisées pour le moment (par exemple, un pair qui quitte le réseau doit le signaler au serveur d'accueil, notamment pour éviter de servir de point d'entrée à un futur arrivant).