Rapport de projet: Blockchain distribuée

Lucas SCHOTT

13 mai 2018

1 Introduction

1.1 Présentation

Le but de ce projet était de réaliser une blockchain distibuée et de observer son évolution en simulant un cas d'utilisation de cette blockchain. Malheurseusement ce projet n'a pas abouti par manque de temps (du aux autre projets et examens que j'ai priorisés), et par un mauvais choix de technologie.

Dans cette blockchain distribuée, les noeuds bloc stockent la blockchain, receptionnent des demandes de transactions et mine des block pour valider ces transactions. Les noeuds participants font des demandes de transactions, et peuvent demander aux noeuds bloc la quantité de points blockchain dont ils disposent.

Ce projet à été réalisé en C++ en utilisant les sockets. Il est uniquement utilisable avec des adresse IPv6.

1.2 fonctionnement

Un participant peut contacter un noeud bloc et lui envoyer une demande de transacation d'un compte a un autre, ce noeud bloc stock met la transaction en attente. Le participant peut demander a un noeud bloc quelle est la quantité de points blockchain dont dispose un compte, c'est ce qu'on appelle la balance elle peut etre négative si un participant effectue une transaction supérieur à ce qu'il dispose.

Les noeud blocs attendent d'avoir suffisement de transactions en attente pour ensuite miner un block et valider les transactions. Il lui faut 10 transactions en attente pour commencer le minage. Une fois le block miné, le noeud bloc va l'envoyer à tous les autres noeuds bloc auquel il est connecté.

Plusieurs noeuds bloc peuvent se connecter entre eux, au moment de la connection de deux blocs ceux-ci s'échangent les blockchain qu'ils possèdent. Le noeud bloc garde la blockchain valide la plus longue. Ensuite les noeuds bloc s'envoient mutuellement les listes des autres noeuds bloc qu'il connaissent, et les noeuds blocs vont ainsi démarrer de nouvelles procédure de connexion avec les autres noeuds bloc. Tous les noeaud blocs sont connectés entre eux avec une topologie full-mesh. Ainsi quand un noeud bloc mine un nouveau block, il va le transmettre à tous les autres noeuds bloc.

Les noeuds bloc connectés s'envoient des messages de manière périodique pour maintenir leur connection, si un noeud bloc ne reçoit plus de messages d'un autre noeud bloc il va le conséderer comme mort et les deux noeuds bloc se déconnecteront. Un noeud bloc est considéré comme mort au bout de 30 secondes sans réponse.

Dans la pratique je n'ai pas réussi transférer une blockchain entière d'un noeud bloc à un

autre, ainsi quand un nouveau noeud bloc, s'inscrit auprès des autres noeuds bloc, il n'arrive jamais à obtenir la blockchain. Et les échanges de block de fonctionneront pas.

Donc dans l'état du projet auquel je suis il n'y a pas de distribution de la blockchain. On peut créer un noeud bloc, des noeuds participants peuvent lui envoyer des transactions, et obtenir leur balance. Mais malheureusement ce noeud bloc fait office de serveur central. On arrive néanmoins a connecter les noeuds blocs entre eux. Ils vont s'envoyer des message de maintient de connexion s'échanger les adresses des autres noeuds bloc, mais n'arrivent pas à s'échanger des données relatives à la blockchain.

2 Utilisation

```
$ ./bloc <compte> <adresse_IPv6|nom_de_domaine> <numero_de_port> [-v]
```

Sert à creer un noeud bloc unique, avec un compte pour récupere les récompenses du à son travail de minage, en lui donnant aussi une adresse IPv6 à utiliser disponible sur la machine courante et un numéro de port à utiliser, ce noeud bloc peut recevoir des demandes de transactions

```
$ ./bloc <compte>  <adresse|nom_de_domaine> <numero_de_port>
<adresse|nom_de_domaine> <numero_de_port> [-v]
```

Permet de creer un noeud bloc comme précédement et de le connecter à un autre noeud bloc déjà existant, en lui donnant en plus de sa propre adresse et numéro de son port, l'adresse (ou le nom de domaine) et le numéro de port du server auquel se connecter. Le nouveau noeud bloc communiquera avec celui déjà en place et intègrera le groupe de noeuds bloc dans lequel le noeud bloc contacté est déjà intégré.

```
$ ./participant <adresse_IPv6|nom_de_domaine> <port>
<comtpe_source> <compte_destination> <montant>[-v]
```

Permet a un participant de contacter un noeud bloc en lui faisant une demande de transaction du compte source au compte destination, du montant donné.

```
$ ./participant  <adresse_IPv6|nom_de_domaine> <port>
balance <compte> [-v]
```

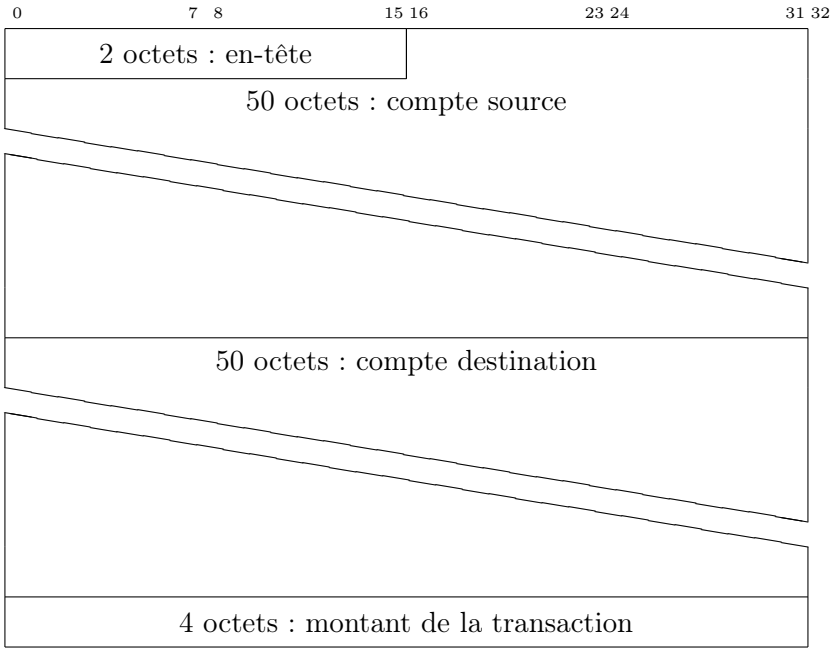
Permet au noeud participant de demander au noeud bloc la balance du compte spécifié(nombre de points blockchain).

Les deux programmes, *participant* comme *bloc* peuvent prendre l'option *-v* (verbose) en argument. Avec cette option activée le programme affiche explicitement ses différentes actions, et plus particulièrement les détails de synchonisations entre server, et les émissions et réceptions de messages.

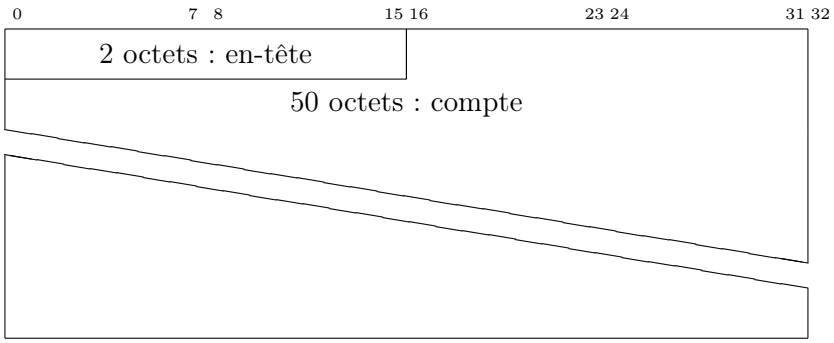
3 Format de messages

Il y a cinq formats de messages :

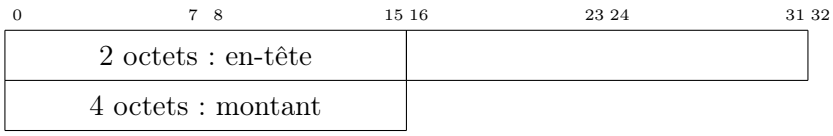
— Le format *transaction_ t* est utilisé quand un participant ou un noeud bloc envoi un transaction à un autre noeud bloc.



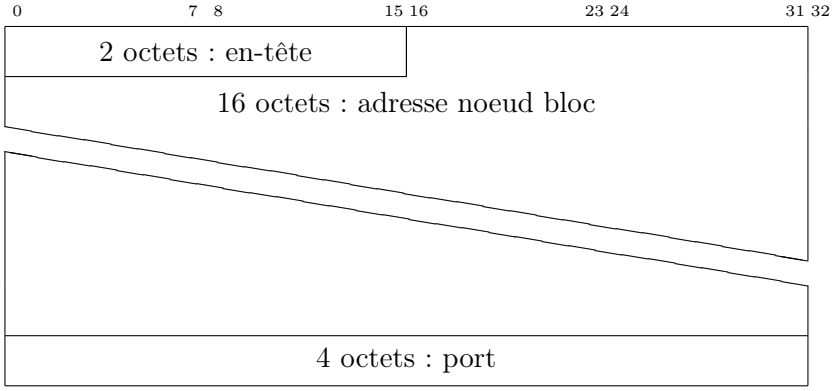
— Le format *ask_ balance_ t* est utilisé quand un participant demande sa balance à un noeud bloc.



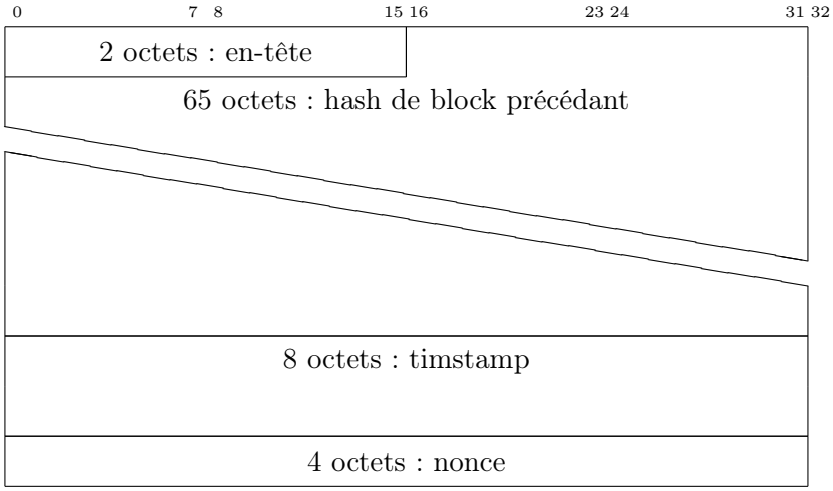
— Le format *send_ balance_ t* est utilisé quand un noeud bloc envoi le montant de la balance d'un participant à celui ci.



— Le format *syn_ t* est utilisé quand un noeud bloc envoi des messages de connexion ou de synchronisation avec un autre noeud bloc.



— Et le format *block_ t* est utilisé quand un noeud bloc envoi un block a un autre noeud bloc.



4 Implémentation

4.1 Échanges client/server

4.1.1 Participant

Algorithme 1 : Client : algorithme principal

```
ouverture de la socket vers le noaeud bloc
si le message est une transactinon alors
    envoyer les données au noeud bloc
sinon si le message est une demande de balance alors
    envoyer la requête au noeud block
    attendre la réponse du noeud bloc pendant une seconde
    afficher la réponse
fin si
```

4.1.2 Noeud bloc

Algorithme 2 : Server : réponse au client

```
tant que le server est ouvert faire
    reception de message
    si le message est une transaction alors
        mettre la transaction en attente
        si il y a suffisament de messages en attente alors
            miner un block
            ajouter la block a la blockchain
            envoyer le block au autres noeud bloc
        fin si
    sinon si le message est balance alors
        parcourir la blockchain
        envoyer le montant au client
    fin si
fin tant que
```

4.2 Connexion des servers

Algorithme 3 : Server thread1 : connexion entre servers

```
tant que le server est ouvert faire
  reception de message
  si le message est server connect alors
    lui envoyer tous les servers auquels on est déjà connecté
    l'enregistrer dans la table des servers connectés et initialiser la date de dernier
    keep_alive
    lui envoyer un message connect
  sinon si le message est keep alors
    mettre a jour la date de dernier keep_alive de ce server
  sinon si le message est share server alors
    ajouter le server dans la table des servers connus et initialiser la date de dernier
    keep_alive
    lui envoyer un message connect
  fin si
fin tant que
```

Algorithme 4 : Server thread2 : maintient des connections avec les servers

```
tant que le server est ouvert faire
  envoyer un message keep_alive à tous les servers connectés
  tant que parcours de chaque server de la table des servers connus faire
    si la date de dernier keep_alive est ancienne de plus de 30 secondes alors
      supprimer le server de la table
    fin si
  fin tant que
  attendre 5 secondes
fin tant que
```

5 Problèmes rencontrés et Conclusion

Ce projet à été très intéressant à réaliser, je m'y suis pris très tard à cause des nombreux autres travaux à réalier pendant ce semestre. Au commencement du projet, j'ai tendé de l'implémenter un java en utilisant la bibliothèque RM, je me suis rapidment confronté à un prolème car je ne comprennais comment envoyer un block, ou encore moins la blockchain entière à un autre noeud. Alors après avoir été resté bloqué de nombreuses heures, je me suis dit que je vais le faire en C++, en utilisant les socket, en m'inspirant du projet que j'avais réaliser le semestre dernier en cours en réseaux. J'ai bien avancer pendant plusieurs jours jusqu'à être bloqué au moment d'envoyer une blockchain entière, où la succession des messages à envoyer, un block après l'autre suivit de toutes les transactions du block, m'ont totalement empêché d'avancer. Je suis suis alors dit que j'avais fait un mauvais choix en passant sur le C++ avec les socket. Je suis repassé sur le projet en java/RMI et c'est à ce moment la que j'ai enfin réaliser comment envoyer des objets simplement en les sérialisant. Mais à ce moment la il était trop tard. Et je n'ai pas eu le temps de finir. J'ai donc fait en quelque sorte deux projets en parallèle, un en C++ avec les socket et un en java/RMI. Le projet le plus aboutit des deux est le projet en C++, c'est celui que j'ai détaillé dans ce rapport.

Le sujet du projet était très intéréssant, mais je n'ai malheureusement pas pu en explorer tous les recoins. Il m'a néanmoins appris énormément sur le programmation distribué et les nombreux avantages des librairies de programmation distribués contrairement aux socket qui sont bien plus complexes à utiliser. Si je devais le refaire maintenant je le ferait d'une toute autre manière. Même si les résultats ne sont pas la je ressort quand même grandi de ce projet.