2020/2021

Projet S4

Rapport de soutenance

Groupe TREG

CAVALIÉ Margaux
LITOUX Pierre
PINGARD Adrien
RUIZ Hugo
VEYRE Thimot

Encadrant : VERNAY Rémi

TABLES DES MATIÈRES

Introduction

Ce cahier des charges présente notre projet de S4, un logiciel dans lequel l'algorithmique occupe une part très importante. Le sujet est libre, et encadré par le docteur en informatique et expert auprès de la cour d'appel de Toulouse Rémi Vernay. Le projet sera développé sous Linux, et codé en langage C.

Ce cahier contient une présentation générale du projet, en plus de la répartition des tâches et du planning détaillé de réalisation. Notre groupe est actuellement constitué de 5 personnes : Cavalié Margaux, Litoux Pierre, Pingard Adrien, Ruiz Hugo et Veyre Thimot. Il est fort probable que Pierre Litoux parte en séjour à l'international, c'est pourquoi nous sommes un groupe de cinq personnes, et non quatre comme le prévoit le sujet.

Pour notre projet, nous avons choisi de recréer la technologie d'une blockchain. Cette technologie est décrite pour la première fois dès 1991 par les chercheurs Stuart Haber et W. Scott Stornetta, qui cherchaient à horodater des documents numériques pour que ceux-ci ne soient jamais antidatés ou altérés. Cette technologie tombera dans l'oubli, et perdra son brevet en 2004, quatre ans avant la création du Bitcoin.

En 2004, un activiste cryptographique, Hal Finney, lance le système appelé "preuve de travail réutilisable", qui permet de conserver le registre de propriété des informations envoyées pour permettre à n'importe quel utilisateur du réseau de vérifier l'exactitude et l'intégrité des données en temps réel. Ce n'est qu'en 2008 que Satoshi Nakamoto fait naître le Bitcoin en reliant ces deux technologies. Il envoya dix Bitcoins à Hal Finney, et offrit cinquante Bitcoins de récompense à celui qui minerait le bloc. Le Bitcoin et la Blockchain comme nous les connaissons sont ainsi créés.

L'objectif de notre projet est de récréer cette technologie. Pour cela, il nous faudra nous intéresser à la cryptographie et à la création d'un réseau qui mettrait en relation tous les différents utilisateurs. Le dernier rendu du projet est prévu pour la semaine du 14 juin 2021, nous avons ainsi trois mois et demi pour porter notre projet à terme.

État de l'art

Fonctionnement de la blockchain

Le terme blockchain (ou chaîne de blocs) désigne la technologie de consensus décentralisé. La blockchain est donc une technologie de stockage et de transmission d'informations, transparente, sécurisée, fonctionnant sans organe central de contrôle.

Toute blockchain publique fonctionne nécessairement avec une monnaie ou un jeton programmable. Les transactions effectuées entre les utilisateurs du réseau sont regroupées par blocs.
Avant d'être approuvé, chaque bloc est validé par les nœuds du réseau (appelés les "mineurs"), selon des techniques qui dépendent du type de blockchain. Dans la blockchain du bitcoin cette technique est appelée le "Proof-of-Work" (preuve de travail), et consiste en la résolution de problèmes algorithmiques.
Une fois le bloc validé, il est horodaté et ajouté à la chaîne de blocs. La transaction est alors visible pour le récepteur et pour l'ensemble du réseau. Au cours du temps, des blocs sont ajoutés à la chaîne. Une blockchain publique est donc comparable à un grand livre comptable public : anonyme et infalsifiable.

A l'inverse, une blockchain privée est contrôlée par une entité totalement centralisée dans le réseau, et les membres participants doivent avoir été acceptés et déclarés par cette entité. Les transactions peuvent être émises par chacun des nœuds, mais elles sont validées et ajoutées à la chaîne par le nœud central autorisé à le faire. Il n'existe aucun mécanisme de consensus, et les règles de fonctionnement sont spécifiques au dispositif et aux accords passés par les membres de la communauté d'utilisateurs.

La blockchain contient l'historique de tous les échanges effectués entre ses utilisateurs depuis sa création. Cette base de données est sécurisée et distribuée, c'est-à-dire qu'elle est partagée par ses différents utilisateurs, sans intermédiaire, ce qui permet à chacun de vérifier la validité de la chaîne.

Ce qui rend la blockchain si fiable et sécurisée, c'est l'ensemble des méthodes qui permettent aux participants du réseau distribué de se mettre d'accord, sans recourir à un tiers de confiance.

Il existe actuellement deux grandes familles de blockchain. Leur différence réside dans la manière dont les blocs sont validés. La première est la validation par "preuve de travail" (proof of work), et la deuxième la validation par "preuve d'enjeu" (proof of stake).

Validation par proof of work

La validation par preuve de travail est la plus simple à mettre en place. En effet, dans ce système les mineurs doivent résoudre un problème mathématiques très complexe, plus le nombre de mineurs augmente, plus cette complexité augmente également, pour valider un bloc. Ce problème dépend du bloc précédent, de cette manière pour modifier un bloc donné dans la chaîne, il faut également recalculer tous les blocs suivants.

Cette méthode de validation à l'inconvénient majeur d'être extrêmement énergivore et pose un gros problème environnemental.

Validation par proof of stake

Le deuxième principe de validation est la preuve d'enjeu. Cette méthode choisit un ou plusieurs mineurs parmi ceux disponibles et ceux-ci valident le bloc sans calcul complexe. Les mineurs sélectionnés peuvent être choisis selon plusieurs critères (leur ancienneté, leur quantité de monnaie, etc). Ce système permet de réduire grandement la consommation énergétique lié au minage mais il a l'inconvénient majeur d'être de ce fait un peu moins sécurisé

Ces méthodes, pour fonctionner à grande échelle, doivent être rémunérées pour que le réseau continue à fonctionner. En effet, les membres du réseau doivent pouvoir rentabiliser les dépenses liées aux vérifications.

Notre projet

Notre cryptomonnaie utilisera deux familles d'acteurs pour fonctionner.
Les premiers sont les nœuds de gestion, qui s'occupent de recevoir les transactions et de créer les blocs. Les seconds sont les nœuds de minages, qui sont appelés par les nœuds de gestion pour résoudre un problème mathématique permettant de sécuriser la blockchain.
Les nœuds de gestion devront intégrer des sécurités permettant d'éviter la création de blocs malicieux.

Le réseau devra être constitué d'au minimum un nœud de gestion et un nœud de minage. Même si cela implique de gros problèmes de sécurité, nous souhaitons que ce soit possible car c'est un projet, et que le réseau ne sera constitué que de quelques ordinateurs de confiance lors de nos tests.

Pour le système de validation, nous avons décidé de nous orienter vers la validation "proof of work", qui nous semble plus simple à implémenter. Étant donné que le projet restera à petite échelle, l'impact environnemental sera minime. Néanmoins, s'il venait à prendre de l'importance, il nous faudrait le faire fonctionner en "proof of stack" pour réduire son impact énergétique.

La blockchain permet de stocker et transmettre des données, mais les utilisateurs de celle-ci auront besoin d'une interface pour réaliser leurs transactions et consulter l'historique des transactions déjà réalisées. Comme pour la blockchain du Bitcoin, il nous faudra un site ou toutes ces informations sont accessibles en quelque clic , il devra donc intégrer une gestion de compte ou chaque utilisateur aura accès à ses données personnelles tel que visualiser le solde du compte et réaliser des transactions avec les autres utilisateurs. Un site avec une certaine sécurité devra alors être mis en place pour éviter que d'autres personnes y accèdent de façon illégitime et détourne nos fonds.

Nous coderons notre site en HTML5 et CSS, puis nous utiliserons le certificat HTTPS pour garantir aux utilisateurs la confidentialité et l'intégrité de leur données lorsqu'ils en envoient ou en reçoivent. Les utilisateurs pourront ainsi créer leur compte et

obtenir un portefeuille qui leur permettra d'échanger des données avec les autres utilisateurs du réseau en toute sécurité.

Planning

Répartition des tâches

	Réseau	Noeud de minage	Noeud de gestion	Site web
Margaux		⊕		+
Pierre	+	+	+	
Adrien			⊕	
Hugo	+			⊕
Thimot	⊕			

Légende :

⊕ → Responsable
\+ → Suppléant

Planning détaillé de réalisation

Première soutenance

Réseau :
- Simulation d'un réseau en multi-thread
- Exécution des différent noeuds sur différents process
- Début de mise en réseau sur plusieurs machines

Site web :
- Ébauche de site web

Noeuds :
- Début du noeud de gestion

- Fonction de hash primaire et minage brute force
- Interaction noeud de minage/noeud de gestion primaire

Deuxième soutenance

Mis à jour

Réseau :

- Réseau fonctionnel

Site web :

- Amélioration de l'esthétique du site
- Création de compte
- Intégration interface de transaction

Noeuds :

- Utilisation du noeud de minage pour valider un bloc
- Intégration de plusieurs transactions par bloc
- Mise en place de liste de transaction en cours

Soutenance finale

Réseau :

- Réseau fiable

Site web :

- Site web esthétiquement abouti
- Système de compte et de connexion sur le site

Noeuds :

- Echange de données avec les autres noeuds pour mettre à jours les transaction et la blockchain

Avancement du projet

Pour cette première soutenance nous avons été amenés à changer quelque peu la répartition des tâches, en fonction des goûts personnels de chacun et de la charge de travail que chaque partie nécessitait.
Ainsi Pierre s'est finalement positionné sur la gestion des nœuds de minage au vue de l'annulation de son départ à l'international, plutôt que d'être disponible comme soutien sur les autres tâches.
Margaux quant à elle a finalement rejoint Adrien sur la gestion des nœuds de gestion, après que nous avons réalisé que cette partie nécessite une charge de travail plus importante que ce que nous pensions.

	Réseau	Noeud de minage	Noeud de gestion	Site web
Margaux			⊕	
Pierre		⊕		
Adrien			⊕	
Hugo				⊕
Thimot	⊕			

Tableau indiquant sur quelles parties du projet chaque membre a travaillé depuis la validation du cahier des charges

Noeud de gestion

Pour cette première soutenance nous nous sommes concentrés sur les nœuds de gestion.
Nous avons commencé par définir deux nouvelles structures : BLOCK et TRANSACTION.
Une structure **TRANSACTION** possède trois variables : sender, receiver et amount.

- "sender" est une chaîne de caractères, correspondant à l'identifiant (le nom) de la personne envoyant l'argent de la transaction.
- "receiver" est une chaîne de caractères, correspondant à l'identifiant (le nom) de la personne recevant l'argent de la transaction.
- "amount" est un entier, correspondant au montant de la transaction.

La taille d'une chaîne de caractères contenant un identifiant (sender et/ou receiver) a été fixée à 20. Plus tard, l'identifiant ne sera plus simplement un nom, mais la valeur de hachage de l'identifiant de l'utilisateur.
Pour l'instant, le montant maximum d'une transaction a été fixé à la taille maximum d'un "integer" sur 32 bits, c'est-à-dire $2^{31} - 1$ tokens.

Une structure **BLOCK** possède trois variables : previusHash, transactions et blockHash.

- "previusHash" est une chaîne d'octets, qui contient la valeur de hachage du bloc précédent.
- "transactions" est une liste de TRANSACTION.
- "blockHash" est une chaîne d'octets, qui contient la valeur de hachage du bloc actuel.

Nous utiliserons la structure BLOCK afin de générer les blocs de notre blockchain.
Pour l'instant, nous avons fixé le nombre de transactions par bloc à dix. Plus tard nous pensons changer ce système, pour qu'un bloc ne soit pas créé toutes les dix transactions, mais toutes les *x* minutes.

Par la suite, nous avons travaillé sur les fonctions de hachage. Pour implémenter ces différentes fonctions il nous a fallu décider de la méthode de hachage que nous souhaitions utiliser. Nous avons opté pour la plus connue : le sha256, qui avait en plus l'avantage d'être sécurisé.

Nous avons récupéré une librairie créée par Brad Conte sur internet, à l'aide de laquelle nous avons donc créé notre propre fonction de hachage.
La fonction, sha256, prend une chaîne de caractère et un buffer en paramètre. La chaîne de caractère correspond au texte à hacher, et c'est dans le buffer que la valeur de hachage du texte sera stockée. Le buffer, tout comme la valeur de hachage trouvée, a une taille prédéfinie, qui sera constante quelque soit la taille du texte. Dans notre cas, cette constante, nommée SHA256_BLOCK_SIZE, est de 32 octets.
Le texte passé en paramètre ne peut qu'être une chaîne de caractères contenant des caractères ASCII strictement supérieurs à zéro.

La première fonction, très courte, est *txsToString*. Elle prend une TRANSACTION et un buffer en paramètre, et transforme les trois variables de la structure en chaîne de caractère, afin de permettre, plus tard, de les envoyer à nos autres fonctions. La chaîne de caractère est stockée dans le buffer donné en paramètre, qui doit avoir une taille suffisante pour la stocker.

La deuxième fonction est *getMerkleHash*, qui prend en paramètre une structure BLOCK, et une liste d'octet de taille SHA256_BLOCK_SIZE.
Les transactions du bloc sont d'abord transformées en chaînes de caractères grâce à *txsToString*. Ces chaînes sont ensuite concaténées, et le tout est haché avec sha256.
De cette façon, il est presque impossible que deux valeurs de hachage soient identiques.
Le hash calculé, qu'on appelle le merkleHash, est stocké dans le buffer donné en paramètre. Cette valeur ne correspond pas à la valeur de hachage finale de notre bloc, mais elle sera utilisée pour calculer cette dernière.

Pour finir, la fonction qui nous permet de calculer le hash définitif d'un bloc est *getHash*, qui prend en paramètre un BLOCK et une liste d'octets de taille SHA256_BLOCK_SIZE, tout comme *getMerkleHash*.
getHash récupère le hash du bloc précédent (stocké dans previusHash), puis calcule le merkleHash du bloc actuel. Pour finir elle concatène ces deux hashs, applique la fonction sha256 sur cette nouvelle chaîne, et stocke le résultat obtenu, qui correspond au Hash final du bloc actuel, dans la chaîne d'octets donnée en paramètre.
Cependant, nous avons rencontré un problème auquel nous ne nous attendions pas en implémentant cette fonction.

La fonction sha256 permet de générer une chaîne d'octets compris entre 0 et 255.
En hexadécimal, chaque octet contient deux caractères hexadécimaux (de 0 à F).
Si on dédouble chaque octet de la chaîne pour avoir sa représentation hexadécimale on aura donc chaque octet compris entre 0 et 15. Or, comme on l'a dit précédemment la chaîne de caractères que prend la fonction sha256 en paramètre ne peut pas contenir de caractère égal à zéro.
Pour palier ce problème nous avons donc dû implémenter une nouvelle fonction : sha256ToAscii. Celle-ci prend en paramètre une chaîne d'octets et un buffer, et stocke dans le buffer la valeur ASCII d'un hash, c'est-à-dire qu'elle transforme chaque octet de la liste en sa correspondance en ASCII.

Par la suite, nous avons créé la structure **BLOCKCHAIN**, qui possède pour l'instant deux variables :

- "blocks", qui pointe vers le premier BLOCK de la blockchain.
- "blocksNumber", un size_t indiquant le nombre de blocs contenus par la blockchain.

Nous aurons besoin d'ajouter de nouvelles variables à l'avenir.
Afin de créer et gérer la blockchain, nous avons implémenté quatre nouvelles fonctions.

Il nous fallait tout d'abord pouvoir récupérer le bloc le plus récent d'une blockchain, alors nous avons créé getlastBlock, qui prend une BLOCKCHAIN en paramètre et retourne son tout dernier bloc.

Ensuite nous avons créé addBlock, une fonction plus longue qui nous permet d'ajouter un bloc à une blockchain, tous deux passés en paramètre, en utilisant notamment getLastBlock afin de remplir la variable previusHash de notre nouveau bloc.

Par la suite, pour pouvoir initialiser notre blockchain, il nous a fallu créer la fonction createGenesis, qui initialise le genesis, c'est-à-dire le premier bloc de la blockchain, la sentinelle, qui permet d'initialiser la liste.
Ce bloc ne contient donc pas de transaction, mais il faut quand même lui attribuer une valeur de hachage pour que le bloc suivant puisse renseigner ce hash dans previusHash. Pour cela nous avons appliqué sha256 à une chaîne de caractère choisie arbitrairement. Les genesis de toutes nos blockchains auront donc le même hash, mais cela ne pose pas de problème. Enfin, nous avons implémenté initBlockchain, qui utilise les trois fonctions précédentes pour créer une nouvelle blockchain, et initialiser sa sentinelle.

Noeud de minage

Le nœud de minage manipule le sha256 afin de hacher les blocs tout en générant une preuve. L'utilisation du sha256 permet d'avoir des hachages sécurisés de taille constante.
La fonction de hachage nous permet dans un premier temps de créer les nouveaux blocs de cette manière :

$$sha256(\ hash\ bloc\ précédent\ +\ index\ +\ transaction\ +\ date)\ =\ nouveau\ hash.$$

Néanmoins, ce système peut se trouver être trop rapide car hacher un bloc ne prend pas particulièrement beaucoup de temps alors que nous voulons éviter la double dépense car si le hachage des blocs est trop rapide alors un utilisateur peut se permettre de dépenser deux fois un même token.

Pour le minage nous devons donc fournir un code capable de hacher un bloc tout en générant une preuve associée afin de ralentir la création de nouveaux blocs, pour cela une fonction brute force hache en boucle jusqu'à ce qu'une preuve soit valide.

Le principe est le suivant pour chaque bloc :

$$sha256(\ hash\ bloc\ précédent\ +\ index\ +\ transaction\ +\ date\ +\ preuve)\ =\ nouveau\ hash.$$

Le nouveau hachage créé doit répondre à certains critères, par exemple il doit finir par 0.
Ainsi nous sommes obligés de tester de nombreuses preuves jusqu'à en trouver une qui valide le critère, la sortie d' une fonction de hachage étant presque impossible à anticiper, la preuve doit être calculée par un algorithme de brute force.

Nous disposons donc également d'un moyen de vérifier qu'une preuve est valide relativement à un bloc en vérifiant rapidement que :
$sha256(hash\ bloc\ précédent\ +\ index\ +\ transaction\ +\ date\ +\ preuve)$ est bien égal à nouveau hash.

Les fonctions de hachage sont aussi capables de s'aligner à un indice de difficulté plus cet indice est élevé plus la preuve sera compliquée à trouver ceci permet d'éviter au réseau de calculer trop vite les blocs et donc d'avoir des preuves trop facile a produire.
La méthode simple pour augmenter la difficulté est de rendre les critères de validité du nouveau hash encore plus spécifiques. Par exemple, si le critère de validité est que le nouveau hachage doit terminer par un zéro, on peut augmenter la difficulté en augmentant le nombre de zéros.

Pour l'instant le nœud de minage hache lui-même ses blocs sans générer de preuves, il va de soit que lorsque la fonction de minage sera opérationnelle le nœud de gestion y fera appel pour créer ses nouveaux blocs.
La fonction de hachage que nous avons implémentée fonctionne donc de la manière décrite ci dessus et est fonctionnelle à quelques détails près :

- Si le réglage de la difficulté est possible, celui-ci est trop abrupt et un meilleur système est envisageable.
- le minage ne se fait que sur un seul thread pour l'instant ce qui pose des problème évident d'optimisation.
- la fonction de hachage accepte certains arguments spécifique néanmoins elle peut faire certains problème de conversions sur des arguments plus généraux
- le nœud minage se doit évidemment de communiquer avec le réseau : nous pensons pour l'instant à recevoir et renvoyer les blocs a miner sous forme de chaînes de caractères sur des filedescriptor mais nous ne sommes pas encore certains de comment les noeuds de minage vont communiquer avec les noeuds de gestion.

Régler ces différents problèmes constitue donc l'objectif pour les prochaines séances.

Site web

Notre site, conçu avec HTML5 et CSS, est pour l'instant très simple. Pour comprendre le fonctionnement de l'HTML et du CSS utilisés, nous avons suivi un cours sur OpenClassroom qui nous a permis de comprendre les fondements de ces langages de balisage.

Nous utilisons le HTML pour gérer la structure, le texte, les images, les téléchargement et les changements de pages. Le CSS quant à lui nous permet d'ajouter des menus déroulants, ajuster les couleurs, la taille et l'emplacement de la police...
Nous avons codé nous même la partie en HTML5, en suivant les instructions données par le cahier des charges sur ce que devait contenir le site. Nous avons téléchargé la mise en forme CSS sur ce site : https://html5up.net.

Le site contient pour l'instant une page d'accueil présentant une présentation des actualités concernant notre projet, un onglet "Notre projet" dans lequel nous présentons notre idée, ses origines, tout en présentant les fondateurs des premières blockchains. Nous avons ensuite implémenté un onglet "Avancement", qui présente l'évolution de notre projet à chaque soutenance. Finalement, un onglet "Téléchargement" est disponible, dirigeant vers une page rappelant les consignes du projet, le cahier des charges, et la version finale du projet en lui-même.

Ce site est pour l'instant relativement simple, mais notre objectif est de lier plus tard le site et la blockchain, afin de pouvoir réaliser des transactions depuis le site.

Nous travaillons également sur un système de gestion de comptes utilisant PHP, HTML et MySQL, l'objectif étant de permettre à l'utilisateur de créer un compte sur le site, pour utiliser la blockchain depuis celui-ci.
Pour cela, il nous faut créer une page où les clients peuvent s'inscrire avec leur adresse mail en créant un mot de passe. Ensuite, il nous faudra utiliser une base de données pour stocker les données de connexion des utilisateurs, afin qu'ils puissent se reconnecter. Nous avons décidé d'utiliser MySQL, qui a l'avantage d'être gratuit et assez simple d'utilisation, de plus il y a une grande quantité de tutoriels disponibles sur internet.

Pour relier les informations fournies par les utilisateurs et la base de données nous utilisons des fichiers codés en PHP, qui nous permettront de produire des pages web dynamiques via un serveur HTTP. Nous n'avons pas implémenté ces différentes fonctionnalités pour le moment car il nous faut un serveur capable de lire le PHP, chose sur laquelle nous travaillons actuellement.

Nous avions prévu au départ d'utiliser GitHub, qui permet de publier un site web, mais nous avons réalisé que cela ne fonctionne qu'avec du HTML, c'est-à-dire des sites statiques, chose qui ne nous convient pas. Nous utiliserons donc un serveur privé, ce qui nous permettra d'implémenter le code en PHP. Ensuite nous aurons à mettre en relation notre blockchain et le site web, nous avons commencé à réfléchir sur ce point mais nous ne sommes pas encore sûrs de comment le réaliser. Nous continuons nos diverses recherches sur la mise en liaison du site et du réseau mais une partie importante du projet se base sur le réseau , sans lequel notre blockchain est inutile , ainsi donc nous centrerons plus nos efforts sur le réseau pour la prochaine soutenance et ensuite chercherons à connecter le site et la blockchain.

Réseau

Dans cette partie nous allons expliquer, comment nous avons pensé le réseau, comment est-ce qu'il fonctionne, et ses possibles évolutions.
Nous avons décidé de réduire les communications entre les serveurs à trois choses. Ils peuvent échanger leurs connaissances du réseau, les transactions non validées et la blockchain. Ce simple échange autour des ces trois données permet de poser les bases de la décentralisation. Le principe théorique peut s'avérer plutôt simple mais l'implémentation est beaucoup plus complexe.

La partie gestion du réseau s'occupera du premier type de communication, les autres seront à la charge du nœud de gestion. Néanmoins, pour cette soutenance, nous ne l'avons pas encore implémenté.

Le réseau se chargera donc d'agrandir le réseau connu en partageant ses connexions avec les autres serveurs et inversement très régulièrement. Si nous avons le temps nous pourrons essayer de mettre un place un système de ping pour savoir si les serveurs sont toujours en ligne.

Pour simplifier la communication entre le serveur, nous avons décidé que la communication entre le nœud de gestion et la gestion du réseau doit se faire au moyen de deux files descriptors, un pour la lecture et l'autre pour l'écriture. Cela permet de réduire grandement la complexité du côté du nœud de gestion.

Le réseau repose sur une liste de clients connectés. Pour chaque client, nous enregistrons plusieurs informations, telle que l'adresse IP, son statut (en cours de connexion, connecté, terminé, erreur...) ou encore les files descriptors sur lequel nous pouvons le contacter.
Cette liste originellement vide est remplie au démarrage par une liste de serveurs auxquels nous devons nous connecter. Par la suite, les nouveaux clients sont ajoutés à cette liste et supprimés s'ils se déconnectent.
Elle permet de connaître à tout moment le statut et diverses informations sur les clients.

De plus, la gestion du réseau est séparée en deux threads majeurs. Un pour accepter les connexions entrantes. L'autre pour maintenir ces connexions ouvertes et transmettre l'information.
Le premier agit de manière classique, il accepte toutes les connexions entrantes sur un port précis. Il enregistre ensuite le client sur la liste avec l'état "Connexion en cours".

Le thread en charge de maintenir les connexions ouvertes, trouve tous les clients "en cours de connexion" et réalise un fork pour chaque client, au sein de celui-ci, deux threads sont générés pour pouvoir lire et écrire en permanence de manière asynchrone, il passe ensuite l'état du client a "connecté". Ces deux threads vont être sujet à modifications, en effet, il est fort probable qu'à terme nous n'ayons plus l'utilité.

La gestion du réseau côté serveur est en passe d'avoir sa première version terminée, en effet une grande partie a été faite, et la correction des bugs est en cours. Cependant, une fois cette partie finie, il faudra mettre un en place une interface permettant au nœud de gestion hébergé sur le même processus de communiquer facilement avec le reste du réseau. Cette première version se verra néanmoins sûrement lourdement modifiée dans l'avenir car étant une première implémentation d'un serveur de ce type, des améliorations notables, en termes de performance, de stabilité et de sécurité pourront sans doute être trouvées. De plus, il se peut que certaines choses implémentées ne soient finalement plus utilisées ou inversement qu'il manque encore certaines fonctionnalités.

Nous avons eu des difficultés à mettre en place cette première version car, il nous a fallu utiliser toutes les notions que nous avons récemment apprises (fork, thread, mutex...). De plus, le correction des bugs risque d'être assez ardu car le processus lance plusieurs threads et se divise également en plusieurs fork. Néanmoins, les premiers tests nous montrent que l'algorithme en lui-même semble fonctionner correctement.

Conclusion

Lorsque nous avons choisi de recréer une blockchain décentralisée, nous ne connaissions que vaguement les principes techniques utilisés pour la faire fonctionner. Depuis la validation du cahier des charges nous avons appris beaucoup, néanmoins ce sujet implique des notions simples, mais difficiles à assimiler. Nous avons tous encore, plus ou moins, des zones d'incompréhensions concernant les solutions à certains problèmes, surtout au niveau de la décentralisation. C'est pour cela que certaines parties pourront être amenées à être profondément retravaillées.

Annexes

Historique des commits sur notre répertoire de travail