Blockchain

Sylvain Conchon

Laboratoire Méthodes Formelles

Université Paris-Saclay, CNRS, ENS Paris-Saclay

sylvain.conchon@universite-paris-saclay.fr

Contenu du cours

Ce cours présente les principaux ingrédients de la technologie Blockchain

Loin d'être exhaustif, il permet de se familiariser avec les briques de base suivantes :

- Structures de chaine de blocs, d'arbres de Merkle
- La programmation distribuée à l'aide de sockets
- La programmation concurrente avec des threads
- Les primitives de hachage et signature cryptographiques.

Évaluation

Au choix (à faire en binôme) :

Programmation d'une mini blockchain (type Bitcoin) complète
(avec une démo)

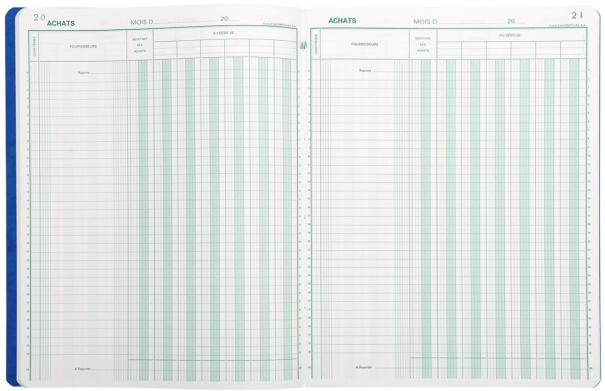
ou

Rapport + Exposé pour approfondir certains aspects vus en cours

INTRODUCTION

Qu’est-ce qu’une Blockchain ?

Une blockchain est comme un grand livre de comptes

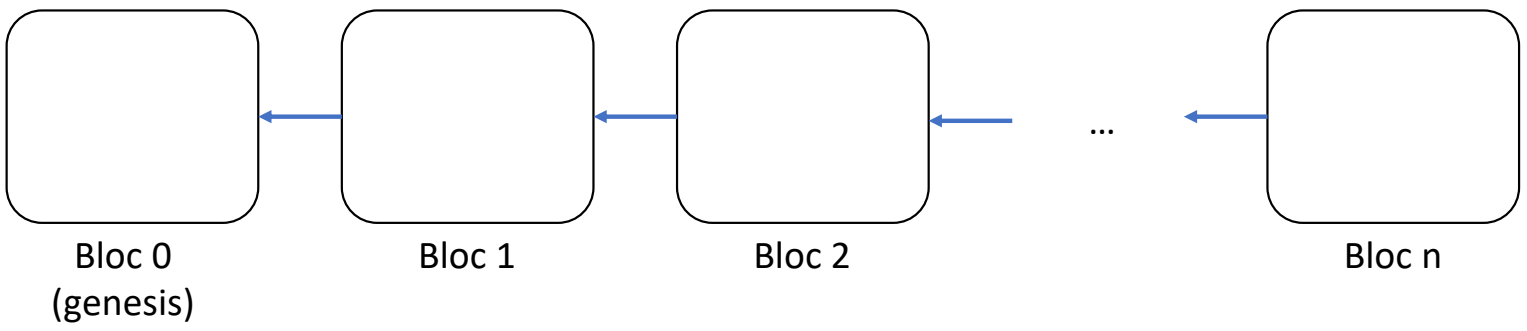


Chaque page de ce livre correspond à un bloc où sont enregistrées des transactions (d’argent ou autre)

Dans le monde des blockchains, ce livre de comptes est appelé un registre (*ledger* en anglais).

Chaînes de blocs

La structure de données qui contient ces blocs est similaire à une liste chaînée. Chaque bloc i contient un pointeur vers le bloc $i - 1$.



Le premier bloc (celui qui commence la blockchain) est traditionnellement nommé genesis.

Transactions

Les transactions enregistrées dans les blocs impliquent deux “clients” :

- celui qui est à l’origine de la transaction
- celui qui est le bénéficiaire de la transaction

Les transactions contiennent d’autres informations, comme une date et un montant

date	de	à	montant
03/01/2021 21:56	Marc	Alain	0.01 Ƀ
03/01/2021 21:58	Marc	Julie	0.02 Ƀ
03/01/2021 21:59	Julie	Bob	0.005 Ƀ
⋮	⋮		

Faire les comptes

À partir de ces transactions contenues dans tous les blocs, on tient à jour les comptes...

Comptes bancaires

Client	Montant
#1	100 Ƀ
#2	20 Ƀ
#3	50 Ƀ
⋮	⋮

Registre distribué

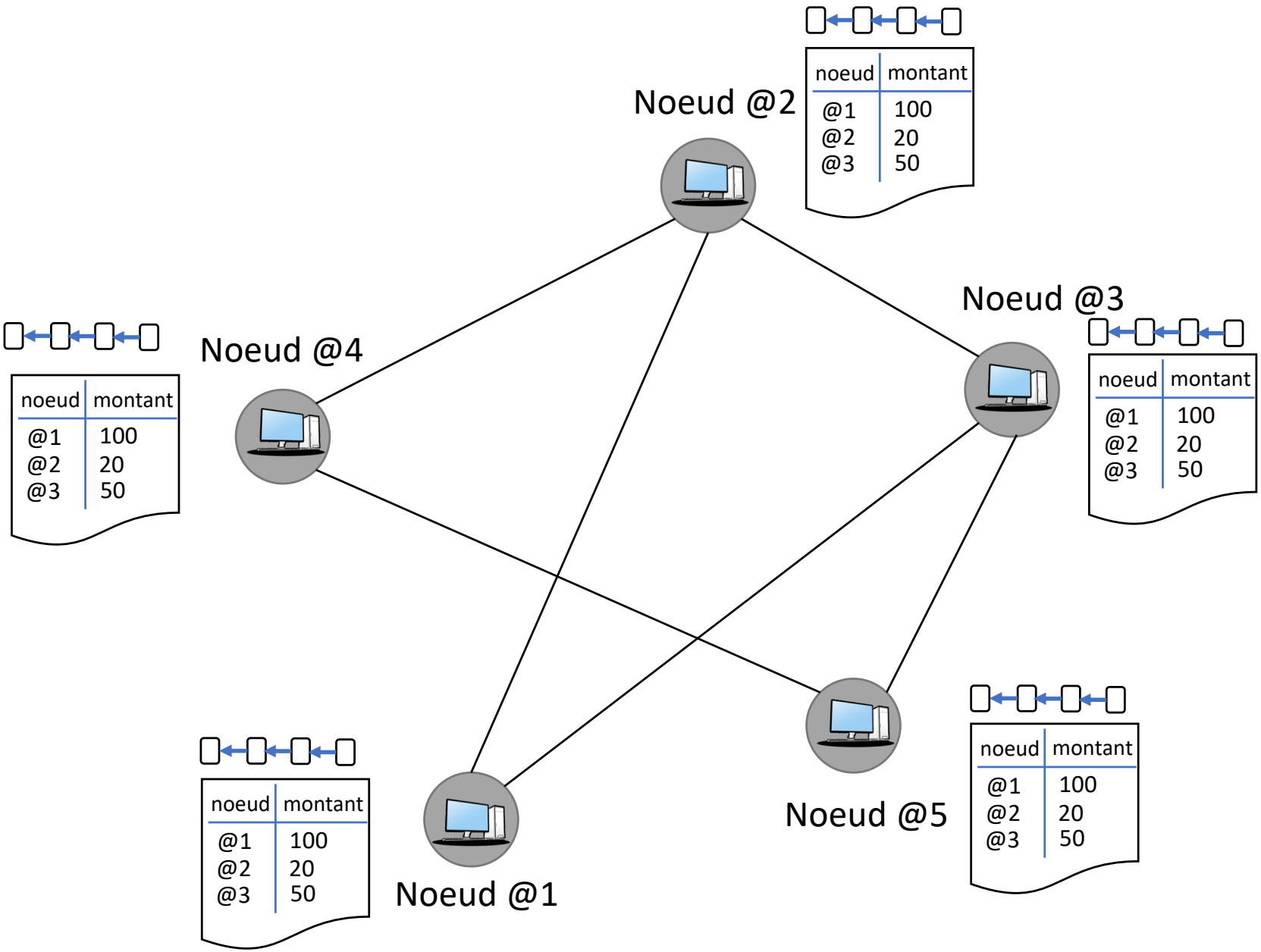
La particularité de la blockchain est que ce registre n'est pas stocké sur un serveur centralisé.

Il est partagé (ou recopié) sur un grand nombre d'ordinateurs à travers le monde entier qui sont tous interconnectés.

On parle de registre distribué, sans aucun organe central de contrôle.

L'intérêt d'une structure distribuée est une très grande résistance aux pannes (des serveurs) et aux attaques de sécurité.

Architecture décentralisée



Quelques problèmes sous-jacents à une blockchain

Comment s'assurer qu'un bloc ne va pas être supprimé de la chaine ? (c'est-à-dire qu'une page du registre ne soit supprimée)

Comment vérifier l'intégrité d'un bloc ? C'est-à-dire que personne ne peut modifier le contenu des transactions ou en ajouter, en supprimer ?

Comment faire pour que toutes les machines aient la même version du registre ?

Comment faire pour être sûr qu'une personne ne dépense pas plus qu'elle n'a d'argent ?

Comment s'assurer qu'une personne ne dépense pas l'argent d'une autre ?

Etc.

La technologie Blockchain

Pour résoudre ces problèmes, l'implémentation d'une blockchain repose (entre autre) sur :

- une architecture de réseau pair-à-pair (P2P)
- un protocole de consensus distribué
- des primitives cryptographiques

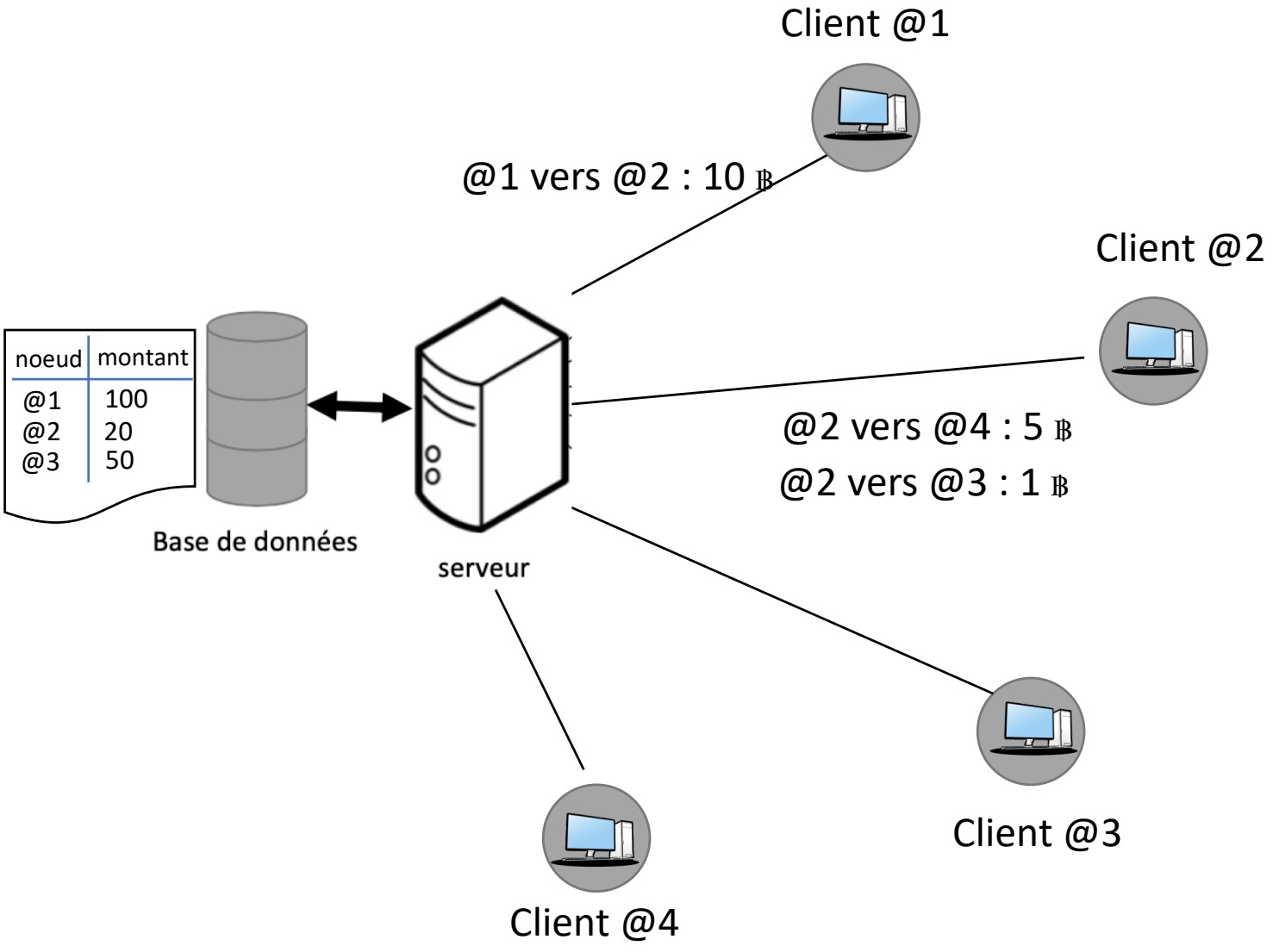
Réseaux P2P

Un réseau Pair-à-Pair (Peer-to-Peer en anglais) est constitué d'un ensemble d'ordinateurs qui s'organisent entre eux pour s'échanger de l'information (fichiers, etc.)

Contrairement à une architecture centralisée du type client/serveur, avec un serveur et des clients qui s'y connectent, les ordinateurs d'un réseau P2P jouent à la fois les rôles de client et de serveur.

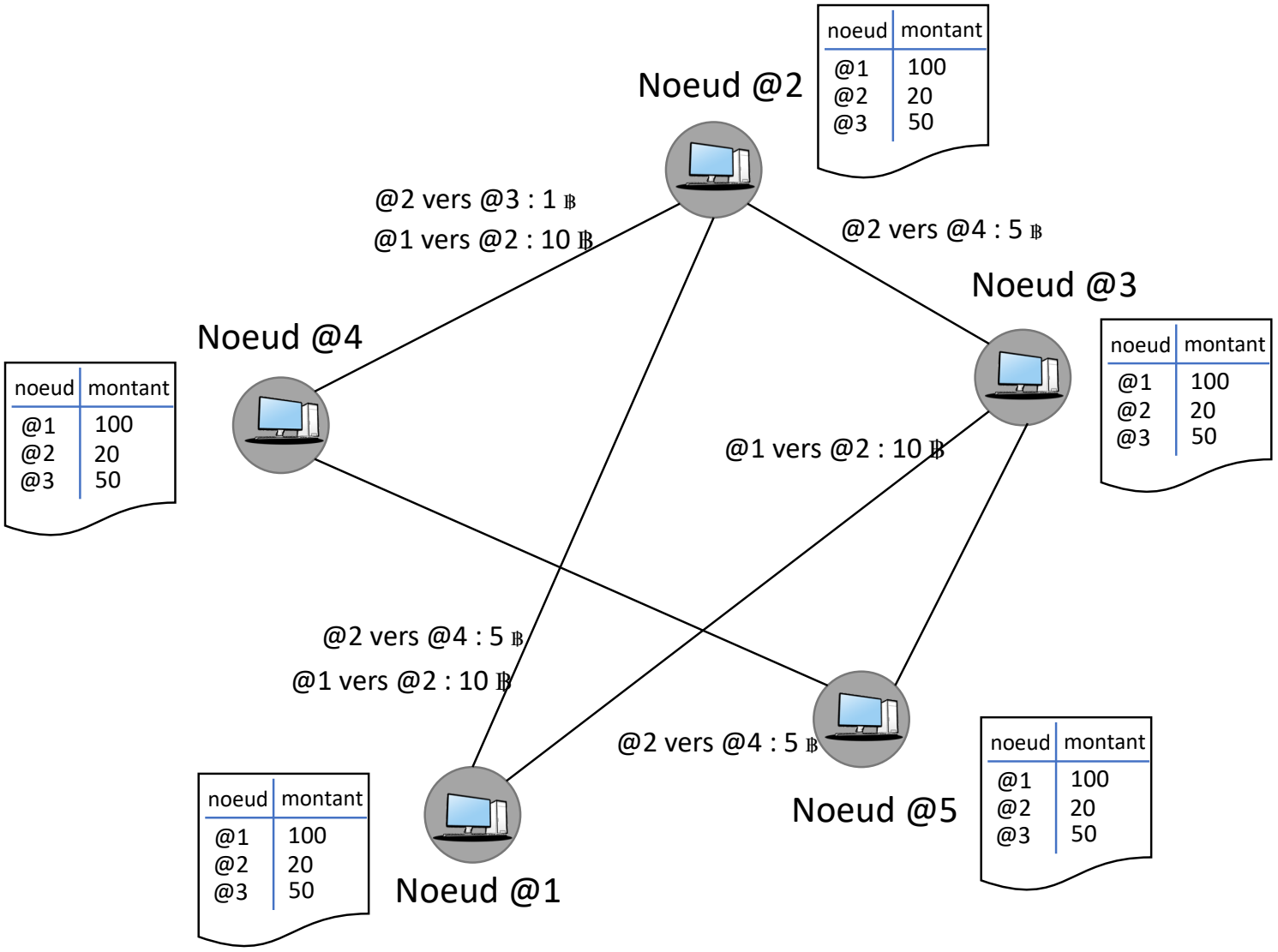
Exemples de réseaux P2P : KazaA, BitTorrent, eDonkey, eMule

Architecture Client-Serveur



Architecture Pair-à-Pair

La distinction entre client et serveur disparait : il n'y a plus que des noeuds qui communiquent directement les uns avec les autres



Avantages des réseaux P2P (vs. Client/Serveur)

Les avantages des réseaux P2P (en comparaison des réseaux Client/Serveur) :

Avantages des réseaux P2P (vs. Client/Serveur)

Les avantages des réseaux P2P (en comparaison des réseaux Client/Serveur) :

- Aucune hiérarchie centrale (réseau maillé) : chaque nœud doit recevoir, envoyer et relayer les données.

Avantages des réseaux P2P (vs. Client/Serveur)

Les avantages des réseaux P2P (en comparaison des réseaux Client/Serveur) :

- Aucune hiérarchie centrale (réseau maillé) : chaque nœud doit recevoir, envoyer et relayer les données.
- Réduction des coûts, simplicité d'utilisation, rapidité d'installation.

Avantages des réseaux P2P (vs. Client/Serveur)

Les avantages des réseaux P2P (en comparaison des réseaux Client/Serveur) :

- Aucune hiérarchie centrale (réseau maillé) : chaque nœud doit recevoir, envoyer et relayer les données.
- Réduction des coûts, simplicité d'utilisation, rapidité d'installation.
- Résistant aux pannes : si des machines tombent en panne, le réseau continue de fonctionner (redondance).

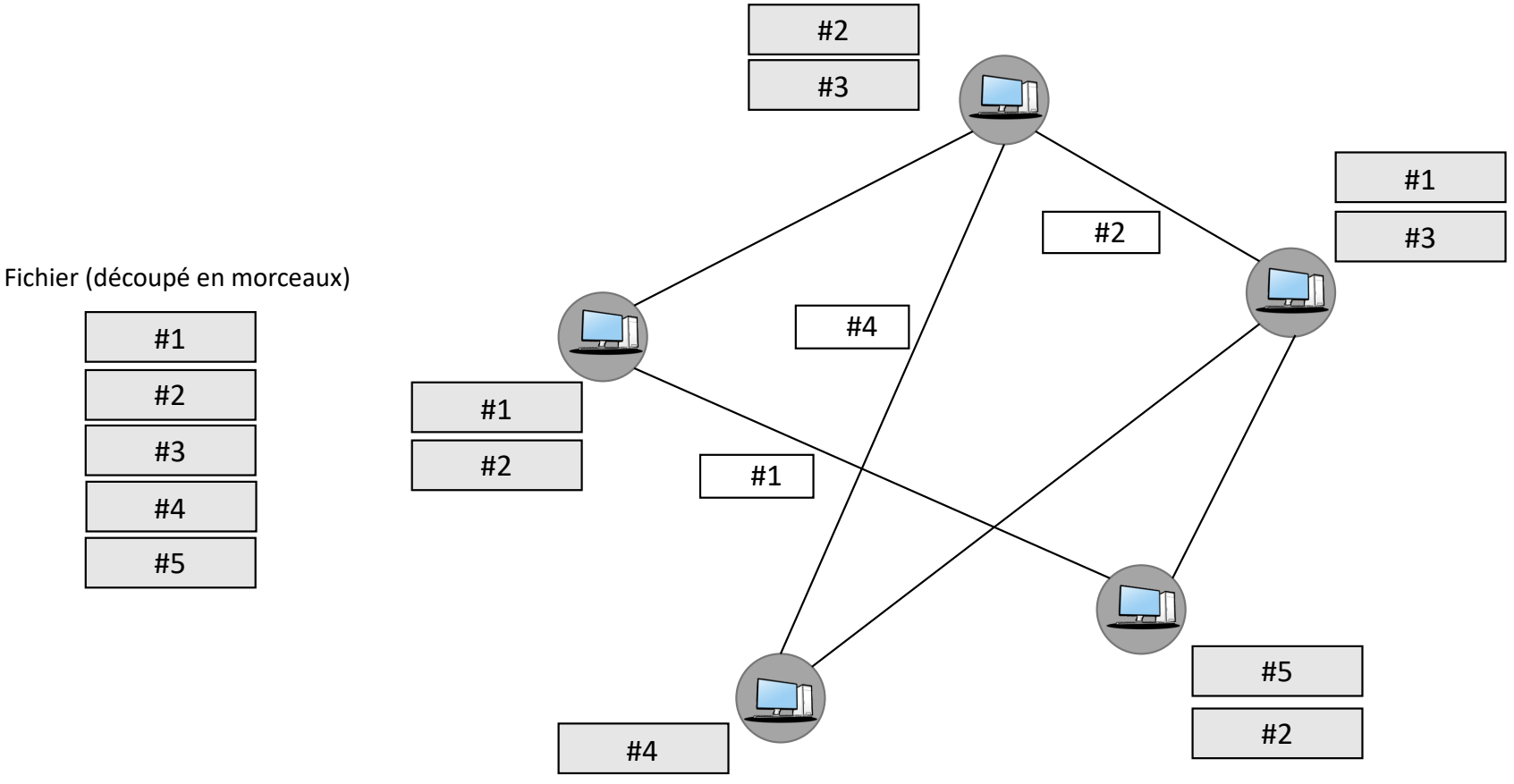
Avantages des réseaux P2P (vs. Client/Serveur)

Les avantages des réseaux P2P (en comparaison des réseaux Client/Serveur) :

- Aucune hiérarchie centrale (réseau maillé) : chaque nœud doit recevoir, envoyer et relayer les données.
- Réduction des coûts, simplicité d'utilisation, rapidité d'installation.
- Résistant aux pannes : si des machines tombent en panne, le réseau continue de fonctionner (redondance).
- Augmentation des performances : plus il y a de machines, plus le réseau est performant

Exemple de réseau P2P : le partage de fichiers

Une des utilisations des réseaux P2P est le partage de fichiers (téléchargement de films ou musiques).



Les fichiers sont découpés en morceaux et les morceaux sont envoyés/récupérés dans un ordre quelconque par les machines.

Inconvénients des réseaux P2P

Les réseaux P2P souffrent principalement de problèmes liés à la sécurité :

Inconvénients des réseaux P2P

Les réseaux P2P souffrent principalement de problèmes liés à la sécurité :

- Intégrité de l'information

Inconvénients des réseaux P2P

Les réseaux P2P souffrent principalement de problèmes liés à la sécurité :

- Intégrité de l'information
- Confidentialité, protection des données

Inconvénients des réseaux P2P

Les réseaux P2P souffrent principalement de problèmes liés à la sécurité :

- Intégrité de l'information
- Confidentialité, protection des données
- Authentification des échanges

Inconvénients des réseaux P2P

Les réseaux P2P souffrent principalement de problèmes liés à la sécurité :

- Intégrité de l'information
- Confidentialité, protection des données
- Authentification des échanges
- Etc.

Algorithmique P2P

- Découverte des autres pairs, gestion des déconnections, pannes, etc.
- Localisation des données sur le réseau
- Routage des messages
- Récupération sur faute, panne

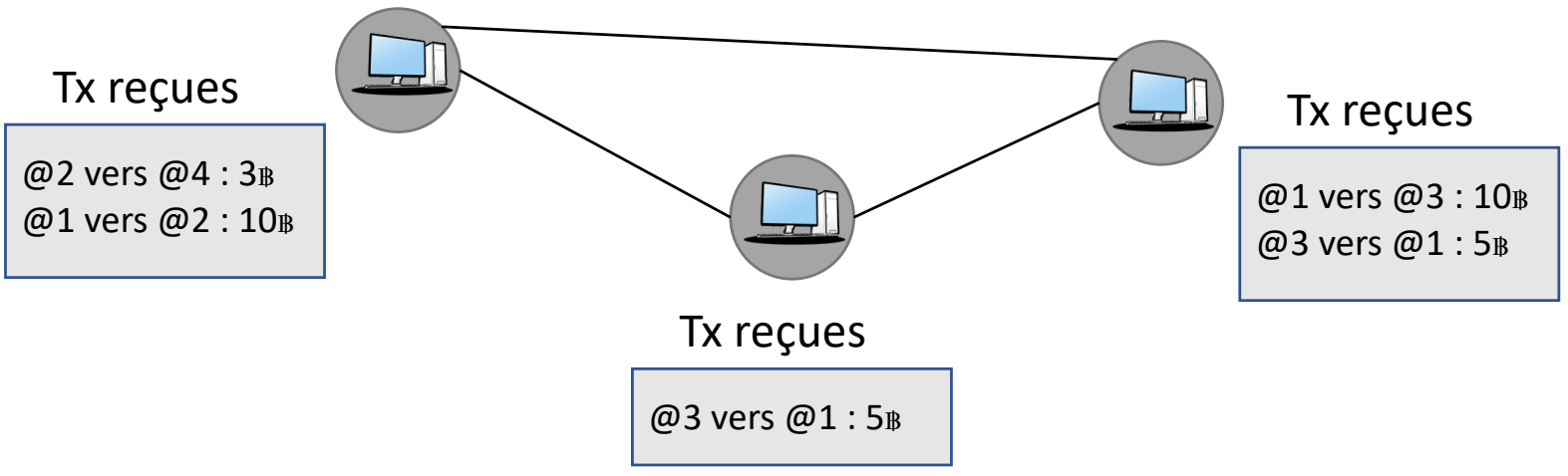
Cele nécessite de mettre en place de protocoles spécifiques entre les pairs.

Voir par exemple :

G. Feltin, G. Doyen, O. Festor. Les protocoles peer-to-peer, leur utilisation et leur détection.
`https://hal.inria.fr/inria-00099498/document`

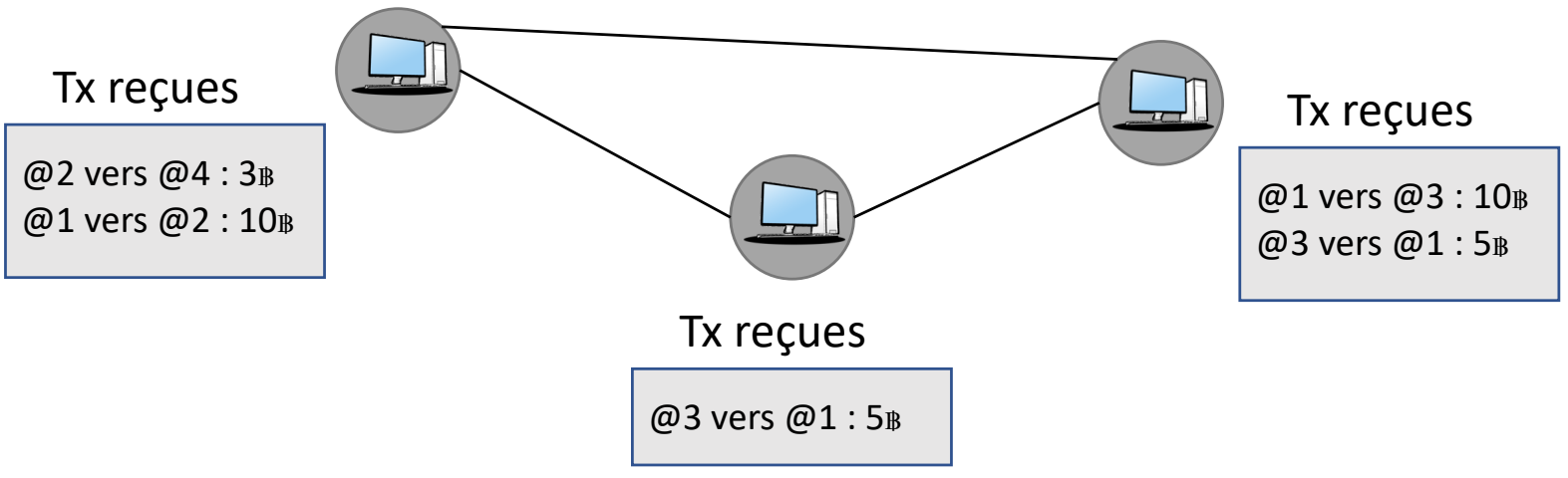
Cohérence dans un réseau P2P

Un réseau P2P ne peut garantir que les nœuds reçoivent les transactions en même temps et dans le même ordre



Cohérence dans un réseau P2P

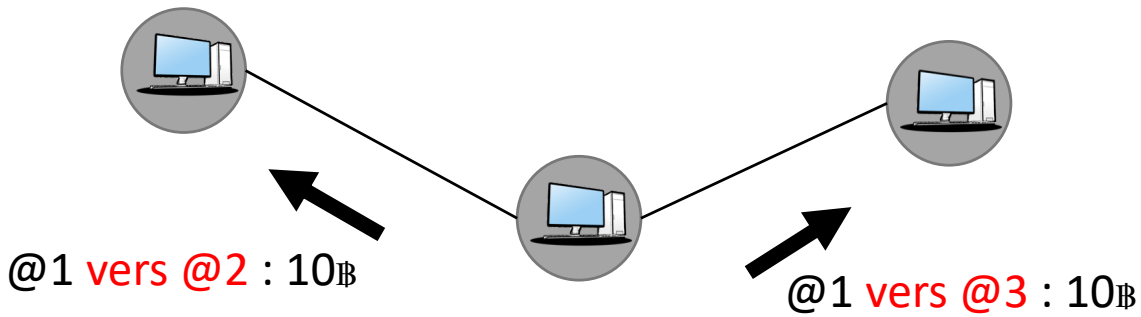
Un réseau P2P ne peut garantir que les nœuds reçoivent les transactions en même temps et dans le même ordre



Comment faire dans ce cas pour que tous les nœuds aient la même version (vision) du livre de comptes ?

Le problème de la double dépense

Dans le cas de la Blockchain, l'incohérence du réseau P2P peut permettre à un client de dépenser plus qu'il ne possède.



clients	฿
@1	10
⋮	

Par exemple, il peut émettre (au moins) deux transactions différentes pour dépenser l'intégralité de son compte et les envoyer à des nœuds différents du réseau.

Consensus distribué

Pour éviter le problème de la double dépense, et plus généralement pour que tous les nœuds d'un réseau aient la même vision du registre, il faut mettre en place un consensus distribué.

La blockchain met en œuvre un protocole de consensus afin que tous les nœuds du réseau s'accordent sur la liste des transactions à valider (i.e. à mettre) dans le prochain bloc de la chaîne.

Le problème du consensus distribué

Soit un réseau constitué de N clients. Chaque client i choisit (au plus) une valeur v_i (au moins un client choisit une valeur).

Le problème du consensus distribué consiste à faire exécuter par les clients un même protocole, de sorte qu'ils communiquent entre eux en s'échangeant des messages et qu'à la fin du protocole ils s'accordent sur une même valeur v (qui doit être l'une des valeurs v_i).

Pannes

Le protocole de consensus doit fonctionner même si certains clients tombent en panne de manière

- permanente : défaillance des machines ou des liens de communication
- temporaire : des messages peuvent être perdus, les clients peuvent être beaucoup trop lents à communiquer, etc.
- byzantine : les clients ne respectent pas les règles du protocole (volontairement ou non).

$\Rightarrow$ Les clients qui ne sont pas en panne doivent obligatoirement terminer.

Résultat d'impossibilité

Théorème FLP [1985, Fischer - Lynch - Patterson]

Dans un système asynchrone (où les messages peuvent être dupliqués, perdus, retardés), le problème du consensus est indécidable dès qu'il peut se produire au moins une panne permanente.

M. J Fischer, N. A. Lynch, M. S. Paterson. Impossibility of distributed consensus with one faulty process.
`https://apps.dtic.mil/sti/pdfs/ADA132503.pdf`

Paxos

En 1989, Leslie Lamport a proposé Paxos, un protocole de consensus pour un réseau asynchrone tolérant aux pannes non-byzantines.

Propriété garantie :

Si moins de la moitié des clients tombent en panne et si le protocole termine, alors tous les processus qui ne sont pas tombés en panne ont la même valeur.

L. Lamport. The Part-Time Parliament.

`http://lamport.azurewebsites.net/pubs/lamport-paxos.pdf`

Byzantin Fault Tolerant

Les protocoles Byzantin Fault Tolerant (BFT) prennent en compte des fautes Byzantines, c'est-à-dire des clients malicieux qui, en plus de tomber en panne, peuvent tenter d'usurper des identités, d'envoyer de faux messages, etc.

Dans le cas d'un réseau synchrone (contrainte de temps de transmission sur les messages) avec N machines, le problème du consensus a une solution (il est sûr et les machines non Byzantines terminent) s'il y a strictement moins de $N/3$ Byzantins.

L. Lamport, R. Shostak, M. Pease. The Byzantine Generals Problem.
`https://lamport.azurewebsites.net/pubs/byz.pdf`

Practical BFT

Algorithme BFT amélioré qui fonctionne pour des réseaux asynchrones.

Utilise des primitives cryptographiques pour signer les messages ou calculer des empreintes numériques.

M. Castro, B. Liskov. Practical Byzantine Fault Tolerance.

`http://cs.brown.edu/courses/cs296-2/papers/bft-nfs.pdf`

PBFT et Blockchains

Un algorithme de consensus dans une blockchain doit également faire face à l'ouverture du réseau :

- Un nombre quelconque de clients qui peuvent s'ajouter ou partir du réseau.

PBFT et Blockchains

Un algorithme de consensus dans une blockchain doit également faire face à l'ouverture du réseau :

- Un nombre quelconque de clients qui peuvent s'ajouter ou partir du réseau.

Comment garantir que le nombre de Byzantins est toujours strictement inférieur au tiers des clients honnêtes ?

PBFT et Blockchains

Un algorithme de consensus dans une blockchain doit également faire face à l'ouverture du réseau :

- Un nombre quelconque de clients qui peuvent s'ajouter ou partir du réseau.

Comment garantir que le nombre de Byzantins est toujours strictement inférieur au tiers des clients honnêtes ?

On ne peut pas, il faut juste faire en sorte que casser ce protocole soit très coûteux et que cela en vaille donc vraiment la peine.

Les protocoles de consensus Blockchain

- Proof of Work (voir un peu plus tard pour les détails)
- Proof of Stake
- Delegated Proof of Stake
- ...

Primitives cryptographiques

L'algorithme de consensus d'une Blockchain, mais également l'intégrité de la chaîne de blocs, reposent principalement sur l'utilisation de deux primitives cryptographiques :

- SHA-2 : famille d'algorithmes de hachage sécurisés (ex. SHA-256 ou SHA-512)
- DSA : algorithme de signature numérique

SHA-256

Il s'agit d'une fonction de hachage `sha256` qui transforme des messages `m` (ou valeurs) de longueur finie (mais quelconque) en des messages condensés `r` de longueur fixe.

SHA-256

Il s'agit d'une fonction de hachage `sha256` qui transforme des messages `m` (ou valeurs) de longueur finie (mais quelconque) en des messages condensés `r` de longueur fixe.

- `sha256(m)` doit être calculé très rapidement

SHA-256

Il s'agit d'une fonction de hachage `sha256` qui transforme des messages `m` (ou valeurs) de longueur finie (mais quelconque) en des messages condensés `r` de longueur fixe.

- `sha256(m)` doit être calculé très rapidement
- `sha256` est à sens unique, c'est-à-dire qu'il est très difficile de calculer `m` à partir de `r`

SHA-256

Il s'agit d'une fonction de hachage `sha256` qui transforme des messages `m` (ou valeurs) de longueur finie (mais quelconque) en des messages condensés `r` de longueur fixe.

- `sha256(m)` doit être calculé très rapidement
- `sha256` est à sens unique, c'est-à-dire qu'il est très difficile de calculer `m` à partir de `r`
- le risque de collisions, c'est-à-dire de messages différents `m1` et `m2` tels que `sha256(m1) = sha256(m2)`, doit être extrèmement faible

SHA-256

Il s'agit d'une fonction de hachage `sha256` qui transforme des messages `m` (ou valeurs) de longueur finie (mais quelconque) en des messages condensés `r` de longueur fixe.

- `sha256(m)` doit être calculé très rapidement
- `sha256` est à sens unique, c'est-à-dire qu'il est très difficile de calculer `m` à partir de `r`
- le risque de collisions, c'est-à-dire de messages différents `m1` et `m2` tels que `sha256(m1) = sha256(m2)`, doit être extrèmement faible
- l'image `r` d'un message `m` doit être très différente de l'image `r'` d'une pertubation très minime de `m`.

SHA-256

Il s'agit d'une fonction de hachage `sha256` qui transforme des messages `m` (ou valeurs) de longueur finie (mais quelconque) en des messages condensés `r` de longueur fixe.

- `sha256(m)` doit être calculé très rapidement
- `sha256` est à sens unique, c'est-à-dire qu'il est très difficile de calculer `m` à partir de `r`
- le risque de collisions, c'est-à-dire de messages différents `m1` et `m2` tels que `sha256(m1) = sha256(m2)`, doit être extrèmement faible
- l'image `r` d'un message `m` doit être très différente de l'image `r'` d'une pertubation très minime de `m`.

SHA-256 produit un haché de 256 bits avec un niveau de sécurité d'une collision pour 2^{128} opérations.

SHA-256

Il s'agit d'une fonction de hachage `sha256` qui transforme des messages `m` (ou valeurs) de longueur finie (mais quelconque) en des messages condensés `r` de longueur fixe.

- `sha256(m)` doit être calculé très rapidement
- `sha256` est à sens unique, c'est-à-dire qu'il est très difficile de calculer `m` à partir de `r`
- le risque de collisions, c'est-à-dire de messages différents `m1` et `m2` tels que `sha256(m1) = sha256(m2)`, doit être extrèmement faible
- l'image `r` d'un message `m` doit être très différente de l'image `r'` d'une pertubation très minime de `m`.

SHA-256 produit un haché de 256 bits avec un niveau de sécurité d'une collision pour 2^{128} opérations.

C'est une version améliorée de SHA-1 (et MD5) qui résiste (pour le moment) aux attaques permettant de créer des collisions sans utiliser un algorithme par force brute (attaque des anniversaires).

DSA

Digital Signature Algorithm (DSA) est un algorithme de signature numérique basé sur la cryptographie asymétrique qui utilise une paire `(pub, priv)` de clé publique et clé privée.

La clé publique peut être donnée à tout le monde, mais la clé privée doit être gardée secrète.

Il s'agit d'une sorte de fonction de hachage qui hache un message `m` en utilisant la clé privée et qui permet de retrouver le message de départ en utilisant la clé publique (ou inversement).

La fonction de hachage `sign(n, priv)` calcule la signature d'un message `m` selon la clé privée `priv`.

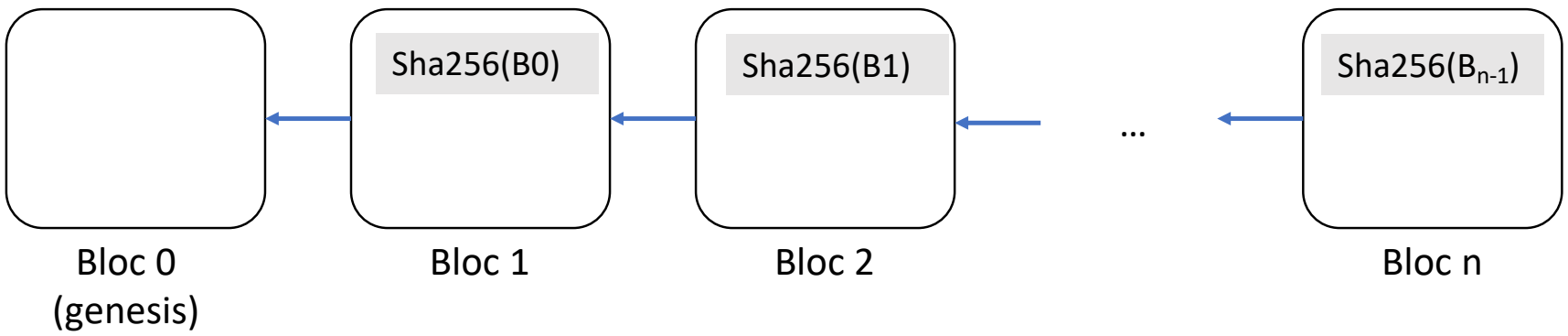
La fonction de hachage inverse `priv(s, pub)` permet de retrouver le message `m` à partir de la signature `s` et de la clé publique `pub`.

```
verify(sign(m, priv), pub) = m
```

Chaîne de blocs sécurisée (1/2)

On utilise la fonction de hachage SHA-256 pour garantir l'intégrité de la chaîne de blocs.

Pour cela, un bloc n contient toujours l'empreinte numérique du bloc $n-1$.



De cette manière, il est impossible d'ajouter ou supprimer un bloc sans que cela ne se voit.

Chaîne de blocs sécurisée (1/2)

Est-ce que l'ajout d'une empreinte numérique suffit à sécuriser la chaîne de blocs ?

Chaîne de blocs sécurisée (1/2)

Est-ce que l'ajout d'une empreinte numérique suffit à sécuriser la chaîne de blocs ?

Une manière de modifier ou supprimer un bloc i est de recalculer l'empreinte numérique dans tous les blocs $k > i$.

Chaîne de blocs sécurisée (1/2)

Est-ce que l'ajout d'une empreinte numérique suffit à sécuriser la chaîne de blocs ?

Une manière de modifier ou supprimer un bloc i est de recalculer l'empreinte numérique dans tous les blocs $k > i$.

Comment peut-on empêcher ce calcul ?

Preuve de travail

Pour résoudre ce problème, certaines blockchains comme Bitcoin imposent une forme particulière à l'empreinte d'un bloc, par exemple qu'elle commence par n chiffres 0, où n est un entier choisi en fonction de la difficulté voulue pour calculer les empreintes :

$$\texttt{sha256(b)} = \underbrace{000...000}_{n} xxxx$$

Preuve de travail

Pour résoudre ce problème, certaines blockchains comme Bitcoin imposent une forme particulière à l'empreinte d'un bloc, par exemple qu'elle commence par n chiffres 0, où n est un entier choisi en fonction de la difficulté voulue pour calculer les empreintes :

$$\texttt{sha256(b)} = \underbrace{000...000}_{n} xxxx$$

Pour résoudre ce problème, seul un nombre entier (appelé nonce) peut être ajouté au bloc.

Il n'existe pas de méthode connue autre que la force brute pour résoudre ce problème. Il faut donc dépenser beaucoup d'énergie (donc d'argent) si on souhaite tricher.

Cette technique de sécurisation de la blockchain est appelée preuve de travail (Proof-of-Work, POW)

Consensus par preuve de travail (1/3)

L'ajout d'un bloc (donc des transactions) dans une blockchain imposant la preuve de travail nécessite que des machines se chargent d'effectuer le (lourd) calcul de l'empreinte du bloc.

Ces machines sont appelées des mineurs.

Consensus par preuve de travail (1/3)

L'ajout d'un bloc (donc des transactions) dans une blockchain imposant la preuve de travail nécessite que des machines se chargent d'effectuer le (lourd) calcul de l'empreinte du bloc.

Ces machines sont appelées des mineurs.

Si plusieurs machines décident de participer à ce calcul, la première qui trouve l'empreinte gagne.

Elle propage alors le bloc qu'elle vient de miner à toutes les autres machines (mineur ou autres), qui peuvent alors vérifier (facilement) qu'elle a en effet trouver la bonne empreinte.

Toutes les machines peuvent alors ajouter ce bloc à leur copie de la blockchain.

Consensus par preuve de travail (2/3)

Malgré la preuve de travail, il est possible que deux machines (ou plus) réussissent à miner un bloc au “même moment”.

Comment choisir quel bloc ajouter à la blockchain ?

Consensus par preuve de travail (2/3)

Malgré la preuve de travail, il est possible que deux machines (ou plus) réussissent à miner un bloc au “même moment”.

Comment choisir quel bloc ajouter à la blockchain ?

Solution :

Supposons qu’une machine ait une copie de la blokchain avec n blocs et qu’elle reçoive un bloc dont l’indice est k.

Consensus par preuve de travail (2/3)

Malgré la preuve de travail, il est possible que deux machines (ou plus) réussissent à miner un bloc au “même moment”.

Comment choisir quel bloc ajouter à la blockchain ?

Solution :

Supposons qu’une machine ait une copie de la blokchain avec n blocs et qu’elle reçoive un bloc dont l’indice est k.

- Si $k = n + 1$, alors elle ajoute ce bloc à sa (copie de la) blockchain, s’il est valide et que les transactions qu’il contient le sont également.

Consensus par preuve de travail (2/3)

Malgré la preuve de travail, il est possible que deux machines (ou plus) réussissent à miner un bloc au “même moment”.

Comment choisir quel bloc ajouter à la blockchain ?

Solution :

Supposons qu’une machine ait une copie de la blokchain avec n blocs et qu’elle reçoive un bloc dont l’indice est k.

- Si $k = n + 1$, alors elle ajoute ce bloc à sa (copie de la) blockchain, s’il est valide et que les transactions qu’il contient le sont également.
- Si $k \leq n$, alors elle l’ignore

Consensus par preuve de travail (2/3)

Malgré la preuve de travail, il est possible que deux machines (ou plus) réussissent à miner un bloc au “même moment”.

Comment choisir quel bloc ajouter à la blockchain ?

Solution :

Supposons qu’une machine ait une copie de la blokchain avec n blocs et qu’elle reçoive un bloc dont l’indice est k.

- Si $k = n + 1$, alors elle ajoute ce bloc à sa (copie de la) blockchain, s’il est valide et que les transactions qu’il contient le sont également.
- Si $k \leq n$, alors elle l’ignore
- Si $k \geq n$, alors elle demande à la machine ayant miné ce bloc sa copie de la blockchain afin de remplacer la sienne.

De cette manière, chaque machine cherche toujours à avoir la plus grande blockchain, car elle représente une preuve d’un travail plus importante, donc un plus grande ”richesse”.

Consensus par preuve de travail (3/3)

Quel intérêt une machine d'une blockchain a-t-elle à miner des blocs, puisque cela coûte très cher ?

Consensus par preuve de travail (3/3)

Quel intérêt une machine d'une blockchain a-t-elle à miner des blocs, puisque cela coûte très cher ?

Réponse : Miner rapporte de l'argent.

Client légers

Une blockchain est constituée de deux types de nœuds : les miners et les wallets.

Les wallets (ou portefeuilles) sont des clients légers qui ne fonctionnent pas comme un nœud du réseau P2P : ils ne cherchent pas à miner des blocs ni à télécharger l'ensemble de la blockchain.

Ils suivent simplement les transactions qu'ils effectuent en interrogeant un serveur de la blockchain.

La technologie sous-jacente à ces wallets est la vérification Simplifiée des Paiements (SPV) qui permet de vérifier qu'une transaction est bien dans une blockchain, sans pour autant télécharger toute la blockchain.

Client légers

Une blockchain est constituée de deux types de nœuds : les miners et les wallets.

Les wallets (ou portefeuilles) sont des clients légers qui ne fonctionnent pas comme un nœud du réseau P2P : ils ne cherchent pas à miner des blocs ni à télécharger l'ensemble de la blockchain.

Ils suivent simplement les transactions qu'ils effectuent en interrogeant un serveur de la blockchain.

La technologie sous-jacente à ces wallets est la vérification Simplifiée des Paiements (SPV) qui permet de vérifier qu'une transaction est bien dans une blockchain, sans pour autant télécharger toute la blockchain.

(voir le cour sur les Arbres de Merkle)

Pour résumer

La blockchain est une structure de données qui implémente un registre, une sorte de base de données transactionnelle, avec les particularités suivantes :

Pour résumer

La blockchain est une structure de données qui implémente un registre, une sorte de base de données transactionnelle, avec les particularités suivantes :

- distribué : le registre est recopié entre plusieurs serveurs

Pour résumer

La blockchain est une structure de données qui implémente un registre, une sorte de base de données transactionnelle, avec les particularités suivantes :

- distribué : le registre est recopié entre plusieurs serveurs
- librement auditable : chacun peut vérifier son intégrité

Pour résumer

La blockchain est une structure de données qui implémente un registre, une sorte de base de données transactionnelle, avec les particularités suivantes :

- distribué : le registre est recopié entre plusieurs serveurs
- librement auditable : chacun peut vérifier son intégrité
- décentralisé : aucune autorité centrale ni tiers de confiance

Pour résumer

La blockchain est une structure de données qui implémente un registre, une sorte de base de données transactionnelle, avec les particularités suivantes :

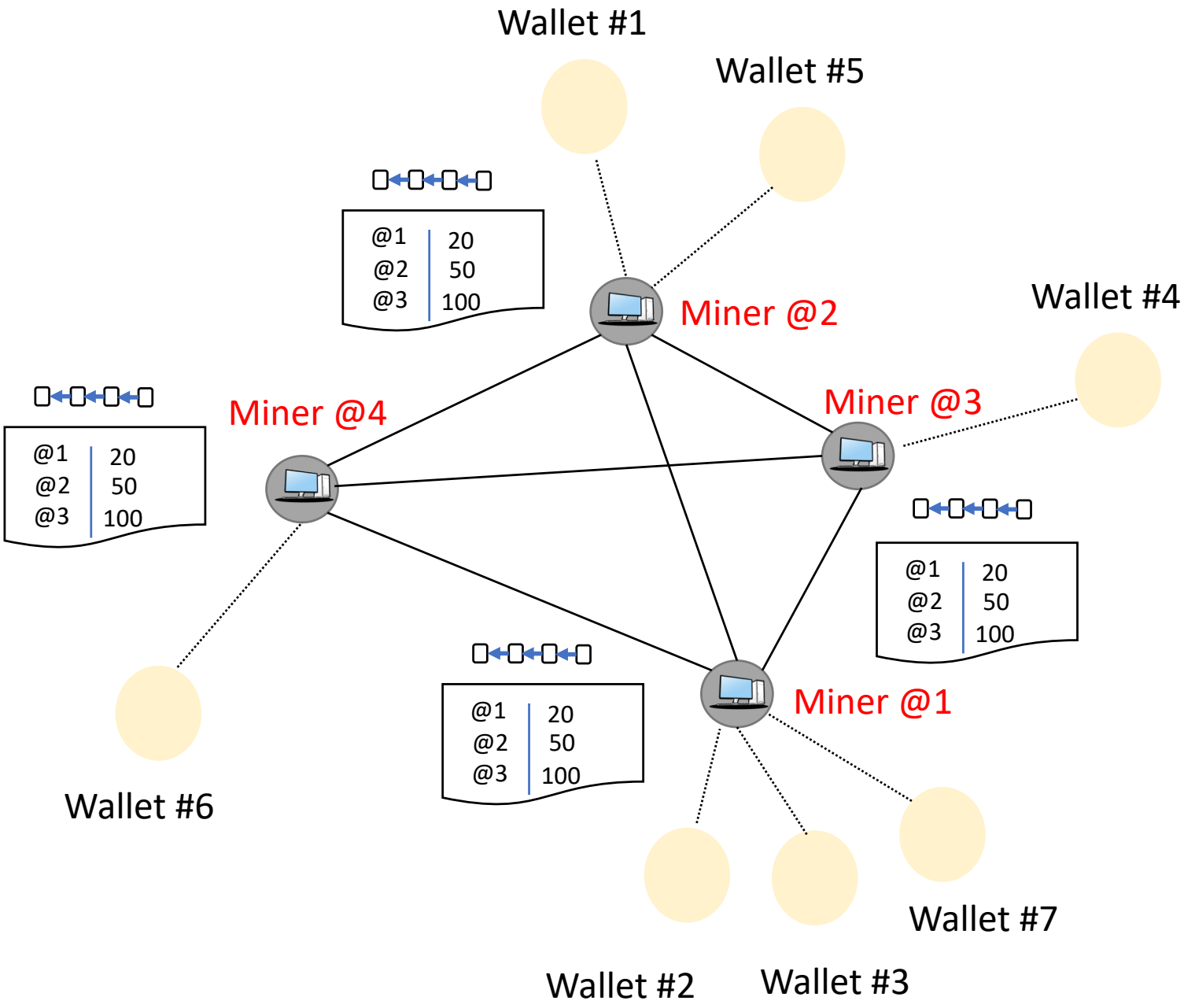
- distribué : le registre est recopié entre plusieurs serveurs
- librement auditable : chacun peut vérifier son intégrité
- décentralisé : aucune autorité centrale ni tiers de confiance
- chronologique : ses transactions sont ordonnées

Pour résumer

La blockchain est une structure de données qui implémente un registre, une sorte de base de données transactionnelle, avec les particularités suivantes :

- distribué : le registre est recopié entre plusieurs serveurs
- librement auditable : chacun peut vérifier son intégrité
- décentralisé : aucune autorité centrale ni tiers de confiance
- chronologique : ses transactions sont ordonnées
- protégé contre les défaillances (pannes), les modifications ou la malveillance (falsification)

Vue d'ensemble



Historique

Quelques faits historiques sur cette technologie (très récente)

Historique

Quelques faits historiques sur cette technologie (très récente)

1. Bitcoin (2009) : la première Blockchain

Historique

Quelques faits historiques sur cette technologie (très récente)

1. Bitcoin (2009) : la première Blockchain

   Aujourd'hui, la capitalisation du marché (Market Cap) de cette cryptomonnaie est de *600 milliards* d'euros environ

   Satoshi Nakamoto. Bitcoin : A peer-to-peer electronic cash system
   `https://git.dhimmel.com/bitcoin-whitepaper/`

Historique

Quelques faits historiques sur cette technologie (très récente)

1. Bitcoin (2009) : la première Blockchain

   Aujourd'hui, la capitalisation du marché (Market Cap) de cette cryptomonnaie est de *600 milliards* d'euros environ

   Satoshi Nakamoto. Bitcoin : A peer-to-peer electronic cash system
   `https://git.dhimmel.com/bitcoin-whitepaper/`

2. Ethereum (2015) : la première Blockchain 2.0 avec smart contracts Turing complets

Historique

Quelques faits historiques sur cette technologie (très récente)

1. Bitcoin (2009) : la première Blockchain

   Aujourd'hui, la capitalisation du marché (Market Cap) de cette cryptomonnaie est de *600 milliards* d'euros environ

   Satoshi Nakamoto. Bitcoin : A peer-to-peer electronic cash system
   `https://git.dhimmel.com/bitcoin-whitepaper/`

2. Ethereum (2015) : la première Blockchain 2.0 avec smart contracts Turing complets

3. Hyperledger Fabric (2015) : initié par "The Linux Foundation", rejoint par IBM, Intel, CISCO, . . .

Historique

Quelques faits historiques sur cette technologie (très récente)

1. Bitcoin (2009) : la première Blockchain

   Aujourd'hui, la capitalisation du marché (Market Cap) de cette cryptomonnaie est de *600 milliards* d'euros environ

   Satoshi Nakamoto. Bitcoin : A peer-to-peer electronic cash system
   `https://git.dhimmel.com/bitcoin-whitepaper/`
2. Ethereum (2015) : la première Blockchain 2.0 avec smart contracts Turing complets
3. Hyperledger Fabric (2015) : initié par "The Linux Foundation", rejoint par IBM, Intel, CISCO, . . .
4. Z-Cash (2016) : une blockchain qui garantie l'anonymat de certaines transactions (*zero-knowledge proof*)

Historique

Quelques faits historiques sur cette technologie (très récente)

1. Bitcoin (2009) : la première Blockchain

   Aujourd'hui, la capitalisation du marché (Market Cap) de cette cryptomonnaie est de *600 milliards* d'euros environ

   Satoshi Nakamoto. Bitcoin : A peer-to-peer electronic cash system
   `https://git.dhimmel.com/bitcoin-whitepaper/`
2. Ethereum (2015) : la première Blockchain 2.0 avec smart contracts Turing complets
3. Hyperledger Fabric (2015) : initié par "The Linux Foundation", rejoint par IBM, Intel, CISCO, . . .
4. Z-Cash (2016) : une blockchain qui garantie l'anonymat de certaines transactions (*zero-knowledge proof*)
5. Tezos (2018) : une blockchain auto-évolutive par un mécanisme de vote intégré.

PROGRAMMATION DISTRIBUÉE AVEC DES SOCKETS

Les prises (sockets)

Il s'agit d'une API pour faire communiquer des processus. Ces prises (sockets en anglais) repose sur :

- Une architecture client/serveur
- Une notion d'adresse réseau : IP + port
- Des appels systèmes Unix
- Des protocoles de communications (essentiellement TCP ou UDP)

API normalisée

Une prise est un mécanisme de communication bidirectionelle entre des processus.

La manipulation des sockets se fait à travers une API (un ensemble de fonctions ou appels systèmes) normalisée.

Cette API est proposé dans de nombreux langages de programmation (C, C++, Java, OCaml, etc.) et sur la plupart des systèmes d'exploitation (Linux, Mac OS, Windows).

Documentations

Dans ce cours, nous allons utiliser le langage OCaml pour programmer une mini Blockchain.

Nous utiliserons donc l'API des sockets OCaml dont la documentation se trouve à l'adresse suivante :

`https://caml.inria.fr/pub/docs/manual-ocaml/libref/Unix.html`

Voici également un lien vers un cours de programmation système en OCaml :

`http://gallium.inria.fr/~remy/camlunix/cours.html#htoc45`

Modèle de communication

Les sockets sont des points de communication que des processus peuvent utiliser pour communiquer, i.e. envoyer ou recevoir des données.

Le schéma pour qu'un processus client communique avec un processus serveur est le suivant :

Modèle de communication

Les sockets sont des points de communication que des processus peuvent utiliser pour communiquer, i.e. envoyer ou recevoir des données.

Le schéma pour qu'un processus client communique avec un processus serveur est le suivant :

1. Le serveur crée une prise s et la connecte à une adresse IP sur un certain port p. Puis il attend que des clients se connectent à cette prise.

Modèle de communication

Les sockets sont des points de communication que des processus peuvent utiliser pour communiquer, i.e. envoyer ou recevoir des données.

Le schéma pour qu'un processus client communique avec un processus serveur est le suivant :

1. Le serveur crée une prise s et la connecte à une adresse IP sur un certain port p. Puis il attend que des clients se connectent à cette prise.
2. Un client crée une prise c et la connecte à l'adresse (IP + port) du serveur.

Modèle de communication

Les sockets sont des points de communication que des processus peuvent utiliser pour communiquer, i.e. envoyer ou recevoir des données.

Le schéma pour qu'un processus client communique avec un processus serveur est le suivant :

1. Le serveur crée une prise s et la connecte à une adresse IP sur un certain port p. Puis il attend que des clients se connectent à cette prise.
2. Un client crée une prise c et la connecte à l'adresse (IP + port) du serveur.
3. La connexion du client a pour effet de créer une nouvelle socket sc chez le serveur de manière à ce que ce dernier puisse communiquer en privé avec le client à travers sc.

Modèle de communication

Les sockets sont des points de communication que des processus peuvent utiliser pour communiquer, i.e. envoyer ou recevoir des données.

Le schéma pour qu'un processus client communique avec un processus serveur est le suivant :

1. Le serveur crée une prise s et la connecte à une adresse IP sur un certain port p. Puis il attend que des clients se connectent à cette prise.
2. Un client crée une prise c et la connecte à l'adresse (IP + port) du serveur.
3. La connexion du client a pour effet de créer une nouvelle socket sc chez le serveur de manière à ce que ce dernier puisse communiquer en privé avec le client à travers sc.
4. La socket s du serveur est toujours disponible et elle peut donc servir à accepter de nouvelles connexions de nouveaux clients.

Appels systèmes

Les fonctions de l'API pour manipuler les sockets se trouvent dans le module `Unix`.

Les principales fonctions sont les suivantes :

Création de la socket : `socket`

Liaison de la socket à une adresse : `bind`

Ouverture du service : `listen`

Attente de connexion : `accept`

Démarrer une connexion : `connect`

Création

Un appel `socket d t p` permet de céer une prise.

La fonction renvoie un descripteur (de type `file_descr`) de la socket.

Création

Un appel `socket d t p` permet de céer une prise.

La fonction renvoie un descripteur (de type `file_descr`) de la socket.

L'argument `d` est le domaine de la socket. Il s'agit de la famille de protocoles pouvant être utilisés. On utilisera principalement `PF_INET` (*Internet domain IPv4*).

Création

Un appel `socket d t p` permet de céer une prise.

La fonction renvoie un descripteur (de type `file_descr`) de la socket.

L'argument `d` est le domaine de la socket. Il s'agit de la famille de protocoles pouvant être utilisés. On utilisera principalement `PF_INET` (*Internet domain IPv4*).

L'argument `t` détermine le type de la socket : `SOCK_STREAM` (pour un mode connecté, soit TCP), `SOCK_DGRAM` (pour un non connecté, soit UDP), `SOCK_RAW` (pour accéder aux protocoles de la couche reseau comme IP)

Création

Un appel `socket d t p` permet de céer une prise.

La fonction renvoie un descripteur (de type `file_descr`) de la socket.

L'argument `d` est le domaine de la socket. Il s'agit de la famille de protocoles pouvant être utilisés. On utilisera principalement `PF_INET` (*Internet domain IPv4*).

L'argument `t` détermine le type de la socket : `SOCK_STREAM` (pour un mode connecté, soit TCP), `SOCK_DGRAM` (pour un non connecté, soit UDP), `SOCK_RAW` (pour accéder aux protocoles de la couche reseau comme IP)

L'argument `p` indique le protocole : on utilisera la valeur 0 pour indiquer le protocole par défaut pour le type de socket sélectionné.

Liaison

Un appel `bind s addr` permet de connecter un descripteur de socket à une adresse.

Liaison

Un appel `bind s addr` permet de connecter un descripteur de socket à une adresse.

L'argument `s` est la socket à connecter.

Liaison

Un appel `bind s addr` permet de connecter un descripteur de socket à une adresse.

L'argument `s` est la socket à connecter.

L'argument `addr` est une adresse représentée par une valeur de type `sock_addr`. Ces adresses peuvent être soit des dans le domaine Unix (`ADDR_UNIX(a)`) soit dans le domaine Internet (`ADDR_INET(ip,port)`).

Liaison

Un appel `bind s addr` permet de connecter un descripteur de socket à une adresse.

L'argument `s` est la socket à connecter.

L'argument `addr` est une adresse représentée par une valeur de type `sock_addr`. Ces adresses peuvent être soit des dans le domaine Unix (`ADDR_UNIX(a)`) soit dans le domaine Internet (`ADDR_INET(ip,port)`).

Pour une adresse Internet `ADDR_INET(ip,port)`, `ip` représente l'adresse IP et `port` le numéro du port (de type `int`)

On peut créer une adresse IP en utilisant `inet_addr_of_string` qui prend en argument une chaîne de caractères contenant l'adresse IP.

```
inet_addr_of_string "192.168.1.5"
```

Ouverture

Un appel `listen s n` permet d'attendre des connexions sur la socket `s` en précisant le nombre maximum `n` de requêtes non encore traitées.

Cela a pour effet que le système enregistre le nouveau service (à cette adresse IP et sur ce numéro de port).

Un appel `lsof -n -i4TCP` dans le terminal permet de voir ce nouveau service.

Attente

Un appel `accept s` bloque le programme en attendant qu'un client se connecte à la socket `s`.

Quand un client se connecte, la fonction renvoie une nouvelle socket pour communiquer avec ce client, ainsi que l'adresse du client.

La socket `s` reste disponible pour d'autres connexions.

Démarrer

Un appel `connect s addr` permet de débuter une connexion sur une socket `s` à une adresse d'un serveur `addr`.

Cet appel et bloquant tant que la connexion ne s'est pas faite.

En interne, l'effet de cette connexion est de brancher la socket `s` à une adresse sur la machine locale (choisie par le système).

Une fois connecté, on peut envoyer ou recevoir des données sur la socket `s`.

Fermer

Un appel `close s` ferme la socket.

Il est également possible de fermer la socket plus finement (en lecture ou écriture) avec `shutdown`.

La désallocation d'une socket peut prendre un peu de temps. Pendant ce temps, il n'est pas possible d'utiliser l'adresse IP et le port de la socket. Pour éviter d'attendre, on peut configurer la socket (après sa création) à l'aide de la fonction `setsockopt`.

Par exemple, `setsockopt s SO_REUSEADDR true` permet la réutilisation immédiate de cette paire (adresse,port).

Canaux de communication

Il existe des fonctions de bas niveau comme `read`, `recv`, `write` ou `send` pour lire ou écrire sur des sockets.

Pour faciliter l'envoi de valeurs sur une socket, on peut créer des canaux de communication par dessus une socket.

Un appel `in_channel_of_descr s` permet de créer un canal d'entrée à partir d'une socket `s`.

De même `out_channel_of_descr sc` crée un canal de sortie.

On peut alors utiliser les fonctions standards d'OCaml pour envoyer ou recevoir des valeurs sur ces canaux, comme par exemple :

```
input_line : in_channel -> string

input_value : in_channel -> 'a

output_string : out_channel -> string -> unit

output_value : out_channel -> 'a -> unit
```

Démo

Exercice 1 : Maillage

On souhaite commencer la couche P2P d'une blockchain. La première étape est de réaliser le maillage du réseau.

On va donc implémenter une (petite) partie d'un *miner*.

Quand un *miner* (A) est démarré depuis un terminal, on indique l'adresse et numéro de port d'un autre *miner* (B) (sauf pour le premier miner à être lancé).

Le miner (A) doit alors se connecter à (B) pour lui indiquer sa présence. En retour, (B) lui envoie la liste M des autres miners qu'il connait, et (B) envoie également les informations de (A) à tous les *miners* de M.

Exercice 2 : Broadcast

En plus d'accepter les connexions des autres *miners*, un *miner* doit aussi accepter les messages envoyés par un *wallet*.

Lorsqu'il reçoit un message M d'un *waller*, le *miner* doit le transmettre à tous les autres *miners* qu'il connait (broadcast).