Ethereum, la prochaine étape des systèmes transparents

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 12 septembre 2015

https://www.bortzmeyer.org/ethereum.html

Je viens de me lancer dans Ethereum et j'ai envie de partager avec vous (partager mon expérience, pas mon argent virtuel, faut pas rêver).

Ethereum est souvent présenté par rapport à Bitcoin : « Bitcoin 2.0 », « "*the next Bitcoin*" », « "*Bitcoin on steroids*" », etc. Il est vrai qu'il s'inspire très fortement de Bitcoin mais, en le présentant ainsi, on peut le confondre avec tous les autres machins qui ont été faits à partir de Bitcoin comme Dogecoin, Litecoin ou encore Namecoin <https://www.bortzmeyer.org/namecoin.html>. À part ce dernier, ces comparaisons peuvent amener à conclure qu'Ethereum n'est qu'« une nouvelle monnaie virtuelle et cryptographique ("*alt coin*") ». Alors qu'il est bien plus que cela.

Ceci dit, c'est vrai que, pour expliquer Ethereum, partir de Bitcoin est pratique. Avant Bitcoin, on considérait que, pour établir et exécuter des contrats (oui, une monnaie est une forme de contrat : je te donne des billets en échange de quelque chose), il n'y avait que deux solutions :

— Une toute petite communauté, composée uniquement de gens honnêtes qui se connaissent,
— Ou bien un organisme de confiance, à qui tout le monde délègue la responsabilité de surveillance : banque centrale dans le cas de la monnaie, registre de noms de domaines pour les noms, etc.

Bitcoin a montré qu'il existait une troisième solution : une structure de données publique, le **livre des opérations** ou "*blockchain*", que tout le monde peut voir et vérifier <https://www.bortzmeyer.org/tousapoil.html>. Cette structure liste toutes les transactions et est protégée par la magie de la cryptographie, de façon à ce qu'elle soit validable. C'est le grand mérite de Bitcoin, et l'invention géniale de Satoshi Nakamoto : prouver théoriquement et expérimentalement qu'un système complètement pair à pair pouvait fonctionner, ce qui était loin d'être évident (par exemple, j'étais personnellement persuadé que c'était impossible).

Bitcoin est spécialisé : on ne peut s'en servir que pour la monnaie. Si on veut, par exemple, enregistrer des noms et non pas échanger de l'argent, il faut copier le code de Bitcoin (il est libre), faire des modifications et créer sa propre infrastructure (sa "*blockchain*" à soi, ses mineurs, ses explorateurs de "*blockchain*", etc). Ce choix n'est pas un oubli ou une erreur des concepteurs de Bitcoin : c'est parce que faire un système équivalent, mais généraliste, est non trivial, notamment du point de vue de la sécurité.

Ethereum reprend plusieurs concepts importants de Bitcoin, notamment la "*blockchain*" et la preuve de travail (il n'est donc pas plus écologiste que Bitcoin). Mais le code est radicalement différent, réécrit de zéro (contrairement à la plupart des "*alt coins*"). Il existe en outre plusieurs implémentations, avec une spécification commune (contrairement à Bitcoin où le code est la spécification, le papier original de Nakamoto ne descendant pas dans les détails). Mais le gros changement par rapport à Bitcoin est que les **transactions** stockées dans la "*blockchain*" ne sont pas limitées à envoyer et recevoir de l'argent. Ethereum dispose d'un quasi-langage de Turing et est donc un système de calcul réparti : les pairs dans le réseau Ethereum ne se contentent pas de vérifier l'intégrité de la "*blockchain*" et d'ajouter de la monnaie, ils exécutent du code arbitraire, celui des applications que vous ou moi développons et envoyons sur le réseau.

Cela permet d'écrire des **contrats** (appelés, dans le style marketing fréquent dans le monde Ethereum, des "*smart contracts*") qui sont la description, dans un langage de programmation, des règles qui s'imposent aux parties contractantes. Un prêt d'argent, par exemple, peut se programmer dans un contrat et s'exécuter automatiquement, sans intervention humaine, et donc sans possibilité de triche. (Question philosophique : quel est le pourcentage des contrats entre humains qui peuvent s'automatiser, c'est-à-dire ne plus accepter d'arrangement, de cas particuliers, etc ?) Les applications d'Ethereum ne sont donc limitées que par votre imagination.

Ces contrats sont la nouveauté importante d'Ethereum. Comme leur exécution peut potentiellement consommer des ressources importantes (imaginez une boucle sans fin dans un contrat...), il faut payer pour leur exécution, ce qu'Ethereum nomme l'**essence** ("*gas*"). Cette essence est payée avec la monnaie de base d'Ethereum, l'**ether**. (D'ailleurs, si vous voulez m'envoyer des ethers, mon adresse actuelle est `0xbe1f2ac71a9703275a4d3ea01a340f378c931740`.) Si vous tombez à court d'essence, l'application s'arrête. C'est cette limite qui fait qu'Ethereum n'est pas une vraie machine de Turing : celle-ci a des ressources infinies.

Bon, vous trouverez des articles généraux et théoriques sur Ethereum un peu partout. Passons plutôt à la pratique. D'abord, un sérieux avertissement, Ethereum est encore vraiment expérimental. Le premier bloc de la "*blockchain*", la **genèse**, n'a été générée que fin juillet 2015 <`https://blog.ethereum.org/2015/07/30/ethereum-launches/`>. Le code ne marche pas toujours, les compétences humaines sont encore rares et, surtout, tout évolue vite et les documentations qu'on trouve en ligne sont presque toujours fausses ou dépassées (ou les deux à la fois). Les interfaces utilisateurs d'accès facile manquent (il n'existe pas encore d'équivalent des nombreux portefeuilles Bitcoin, par exemple). Contrairement à Bitcoin, on n'est donc pas encore en production. Ne vous plaignez pas qu'on ne vous a pas prévenus ! Et, comme avec Bitcoin, vous êtes entièrement responsable de votre sécurité <`https://github.com/ethereum/go-ethereum/wiki/Managing-your-accounts`>. Un bon exemple d'erreur de sécurité faite par un utilisateur et de ses conséquences a été raconté sur Reddit <`https://www.reddit.com/r/ethereum/comments/3ird55/holy_shit_my_eth_accounts_been_hacked/`>.

Donc, installons un nœud Ethereum pour commencer. J'ai dit que, contrairement à Bitcoin, Ethereum n'est heureusement pas défini par une seule implémentation. Deux sont souvent citées pour l'usage pratique d'Ethereum, eth (écrit en C++) et geth (écrit en Go). Essayons avec geth (je dirais plus loin pourquoi eth ne m'a pas satisfait). La voie officielle pour installer un exécutable binaire de geth sur une Debian est :

```
% curl https://install-geth.ethereum.org -L > install-geth.sh
% sudo bash install-geth.sh
```

Cela ne fonctionne pas (rappelez-vous ce que j'ai dit sur le côté expérimental de la chose). Contrairement à ce que prétend la documentation, le code ne marche que sur Ubuntu et il faut, pour Debian, modifier la liste des sources apt <https://www.reddit.com/r/ethereum/comments/3fzatx/cannot_install_ethgeth_on_debian/cttdjv1>. J'édite donc /etc/apt/sources.list.d/ethereum-ethere et je remplace « jessie » (Debian) par « vivid » (Ubuntu). Après, cela fonctionne. Si vous n'aimez pas installer du binaire sans comprendre, le source est disponible en ligne <https://github.com/ethereum/>.

Une fois geth installé, on le lance :

```
% geth console
...
I0912 15:05:24.430701    6306 chain_manager.go:237] Last block (#223283) f374ff2948430d05acc4a14684924d78d6ade38
...
I0912 15:05:25.966841    6306 backend.go:557] Server started
...
instance: Geth/v1.1.3/linux/go1.5
>
```

(Le dernier caractère, le > est l'invite de la console Ethereum.) La console est, par défaut, très bavarde. Le moyen le plus simple de travailler tranquillement est de lancer un autre terminal :

```
% geth attach
```

(Vous pouvez avoir toutes les options de geth en tapant `geth --help` ou bien en ligne <https://github.com/ethereum/go-ethereum/wiki/Command-Line-Options>.) Au premier lancement ("*WARNING : No etherbase set and no accounts found as default*" et "*WARNING : Wrote default ethereum genesis block*"), vous en aurez pour plusieurs heures (en septembre 2015) avant que votre nœud Ethereum soit synchronisé avec la "*blockchain*". (eth est bien plus long que geth pour cette tâche, et il passe en outre du temps à nous prévenir qu'il construit un énorme DAG, nécessaire pour le minage.) À noter que la chaîne de blocs ne fait que s'allonger et, en mars 2016, ce temps était passé à 36 h, sur le même PC... En attendant cette synchronisation, vos transactions ne seront pas visibles localement, puisque votre nœud sera encore dans le passé de la "*blockchain*". Pour voir où en est cette synchronisation :

```
> eth.blockNumber
223298
```

Et vous allez voir sur un explorateur public de la "*blockchain*" si vous vous en approchez. Par exemple, si EtherChain <https://etherchain.org/> ou Eth status <https://eth-status.org/> me disent sur leur page d'accueil qu'on vient de faire le bloc 223299, j'en déduis que je suis quasi-synchronisé (un seul bloc de retard).

La syntaxe de la ligne de commandes de geth peut paraître bizarre mais elle vient du fait qu'elle s'inspire de l'API JavaScript accessible aux programmes Ethereum (et cette API est documentée <https://github.com/ethereum/wiki/wiki/JavaScript-API>, c'est ainsi que vous pouvez apprendre à utiliser la console). Vous pouvez donc, non seulement taper des commandes, mais aussi écrire du JavaScript dans la console. Une documentation plus complète (mais pas toujours correcte) est disponible sous le nom d'« Ethereum Frontier Guide » <http://ethereum.gitbooks.io/frontier-guide/content/>. Il y a aussi une bonne documentation de la console interactive <https://github.com/ethereum/guide/blob/master/interactive_console.md> (pensez juste à remplacer `web3.eth.` par `eth.`).

Bien, maintenant que geth est lancé et a fini par se synchroniser (rappelez-vous, plusieurs heures, la première fois), on peut se créer un compte, pour envoyer et recevoir des ethers :

```
> personal.newAccount("Tu ne sauras pas mon mot de passe")
"0xbe1f2ac71a9703275a4d3ea01a340f378c931740"

> eth.getBalance(eth.accounts[0])
0
```

Il peut y avoir plusieurs comptes et `eth.accounts[0]` est le premier créé. Son adresse est `0xbe1f2ac71a97032` C'est là où vous pouvez m'envoyer des ethers. Puisqu'Ethereum, comme Bitcoin, est transparent, vous pouvez regardez sur un explorateur public toute l'activité de ce compte <https://etherchain.org/account/0xbe1f2ac71a9703275a4d3ea01a340f378c931740>.

La commande `getBalance` indiquait mon niveau de pauvreté. Aucun ether disponible. Je pourrais en miner en faisant tourner mon CPU (et donc mes ventilateurs) à fond mais j'ai préféré en acheter des tout faits, chez Kraken <https://www.bortzmeyer.org/bitcoin-marches.html> où on peut désormais acheter et vendre des ethers. Menu "*Trade / New order*", j'ai acheté deux ethers. Attention, les interfaces Ethereum comptent parfois en ethers mais parfois aussi dans leurs subdivisions, portant des noms pittoresque comme « szabo » (un millionième d'ether) ou « lovelace » (un millionième de milliardième d'ether). Une fois mes ethers obtenus, je les envoie depuis Kraken vers mon compte, en indiquant l'adresse de celui-ci (menu "*Funding / Withdraw*"). Attention, si vous utilisez Kraken pour d'autres monnaies que l'ether, la fonction permettant d'enregistrer une adresse est par compte Kraken et pas par monnaie. Si vous avez enregistré une adresse Bitcoin sous le nom « Maison », vous ne pourrez pas nommer l'adresse Ethereum de la même façon (« "*duplicate withdrawal information*" » est le peu utile message d'erreur de Kraken).

Rapidement, les ethers se retrouvent sur votre compte :

```
> eth.getBalance(eth.accounts[0])
1995000000000000000
```

(Le montant est indiqué en weis, voir plus haut l'avertissement sur les subdivisions de l'ether. Ici, cela fait 1,995 ethers soit même pas deux euros au cours actuel.) Notez bien qu'Ethereum est aussi transparent que Bitcoin : tout le monde peut voir les retraits effectués depuis Kraken (essayez avec un autre explorateur public <http://explorer.etherapps.info/address/0x2910543af39aba0cd09dbb2d50200b3e800a Blockchain).

Jusqu'à présent, rien d'extraordinaire, on a fait exactement la même chose qu'avec Bitcoin. Quel intérêt d'utiliser Ethereum ? Le client est plus perfectionné et permet de faire du JavaScript pour automatiser certaines tâches. Par exemple, ce code :

```
function cab() {
    var i =0;
    eth.accounts.forEach(function(e){
     console.log("  eth.accounts["+i+"]: " +  e + " \tbalance: " + web3.fromWei(eth.getBalance(e), "ether")
i++;
    })
};
```

va itérer sur tous vos comptes et afficher le nombre d'ethers :

```
> cab()
  eth.accounts[0]: 0xbe1f2ac71a9703275a4d3ea01a340f378c931740  balance: 1.9833824 ether
undefined
```

Mais la vraie puissance d'Ethereum n'est pas là. Elle est dans les contrats. Ceux-ci sont des programmes exécutés par la machine Ethereum. Celle-ci exécute un langage machine nommé EVM. Il est de trop bas niveau pour le programmeur normal, qui écrit en général ses contrats dans un langage de plus haut niveau qu'il compilera. Le plus répandu de ces langages est Solidity <https://github.com/ethereum/wiki/wiki/The-Solidity-Programming-Language> (qui ressemble à JavaScript) mais on trouve aussi Serpent <https://github.com/ethereum/go-ethereum/wiki/Managing-your-acco (inspiré de Python) ou LLL (dont la syntaxe ressemble à Lisp mais qui est en fait un langage de bas niveau, très proche du langage machine, Serpent peut d'ailleurs produire du LLL). Comme je suis paresseux, je vais utiliser un contrat en Solidity que j'ai récupéré chez Ethereum.org <https://www.ethereum.org/greeter>. C'est le "*Hello, World*" des contrats. Voyons d'abord si on a bien un compilateur Solidity :

```
> eth.getCompilers()
[""]
```

Non, rien. Installons-le (le script d'installation a mis la bonne source dans ma configuration d'apt) :

```
% sudo apt-get install solc

... (Retour à geth)

> admin.setSolc("/usr/bin/solc")
"solc v0.1.1\nSolidity Compiler: /usr/bin/solc\n"

> eth.getCompilers()
["Solidity"]
```

On peut alors rédiger le contrat (je ne le recopie pas ici, il est en ligne <https://www.ethereum.org/greeter>) :

```
...
(Appel à greeterContract.new()...)
...
Unlock account be1f2ac71a9703275a4d3ea01a340f378c931740
Passphrase:
Contract transaction send: TransactionHash: 0x07dc92c133a433ad58946a7acf8a6f3ccf0352f34882158cc4745c17636ee81e w
undefined
```

La demande de phrase de passe est due au fait que la création du contrat et son exécution nécessitent de l'essence. Cette transaction <https://etherchain.org/tx/0x07dc92c133a433ad58946a7acf8a6f3ccf0352f a nécessité plus de 200 000 unités d'essence et m'a couté en tout 0,0116 ethers, ce que reflète mon portefeuille :

```
> eth.getBalance(eth.accounts[0])
1983382400000000000
```

Une fois validée par les mineurs, le contrat a une adresse :

```
> Contract mined! Address: 0xf0b64c321e9db6bf9164eae8be44443e1e2834a5
[object Object]
```

On peut voir le contrat en ligne <https://etherchain.org/account/0xf0b64c321e9db6bf9164eae8be4 récupérer le code compilé :

```
> greeter.address;
"0xf0b64c321e9db6bf9164eae8be44443e1e2834a5"

> eth.getCode(greeter.address)
"0x60606040526000357c0100000..."
```

Et, bien sûr, l'exécuter :

```
> greeter.greet();
"Mon contrat à moi"
```

La méthode créée par le code est visible dans la définition de l'ABI, ce que les autres utilisateurs devront connaître pour interagir avec « mes » contrats :

```
> greeterCompiled.greeter.info.abiDefinition;
[ ...
{
    constant: true,
    inputs: [],
    name: "greet",
    outputs: [{
        name: "",
        type: "string"
    }],
    type: "function"
    ...
```

C'est évidemment un tout petit contrat, sans grand intérêt. Au fur et à mesure que je progresse en programmation, j'espère en faire des meilleurs. En attendant, il existe plein d'exemples en ligne, comme le DAO <https://www.ethereum.org/dao> qui prétend outrageusement implémenter... la démocratie en Solidity et clame même « *"This is exactly how a democracy should work."* ». Ce contrat met en œuvre un mécanisme de vote entre « actionnaires » sur l'allocation de ressources. Ce vote est censitaire « *"The rules of your organization are very simple : anyone with at least one token [la monnaie du DAO] can create proposals to send funds from the country's account. After a week of debate and votes, if it has received votes worth a total of 100 tokens or more and has more approvals than rejections, the funds will be sent. If the quorum hasn't been met or it ends on a tie, then voting is kept until it's resolved. Otherwise, the proposal is locked and kept for historical purposes."* ». Cela peut être une façon intéressante de gérer certaines organisations mais appeler ça « démocratie » est franchement fort de café. Inutile de dire que je ne partage pas leur délire libertarien.

Un exemple plus simple de contrat, facile à étudier mais illustrant la plupart des possibilités d'Ethereum, est la Pyramide, décrite dans mon article suivant <https://www.bortzmeyer.org/contrat-ethereum.html>.

Revenons à l'autre mise en œuvre d'Ethereum, eth, écrite en C++. C'est en fait par celle là que j'avais commencé mais j'ai arrêté car je n'arrivais pas à lui faire accepter mes contrats. L'installation ressemble beaucoup à celle de geth :

```
% curl https://install-eth.ethereum.org -L > install-eth.txt
%  sudo bash -x install-eth.txt
```

On lance eth, par exemple, ainsi :

```
%  eth -j --frontier -i --verbosity 2
```

Et sa console n'a pas l'air de comprendre le Contrôle-D de fin de session, il faut faire `web3.admin.eth.exit()`. Pour de l'aide, on peut faire `eth --help` (ou bien voir la doc en ligne <https://github.com/ethereum/cpp-ethereum/wiki/Using-Ethereum-CLI-Client>) et juste taper `web3` dans la console.

J'ai créé plusieurs comptes avec eth (l'objet de référence est `web3.eth`, et pas `eth` comme c'était le cas avec geth) :

```
> web3.eth.accounts
['0x003653c3b972ede82f621ac322c6e430493eeb8c', '0x0007ca35a2680425974312233a59a74c00d3c040', '0x00740378b6046b19

> web3.eth.getBalance(web3.eth.accounts[1])
'1995000000000000000'
```

J'avais ensuite essayé de créer un contrat depuis eth mais sans succès. eth a tous les compilateurs :

```
> web3.eth.getCompilers()
['lll', 'solidity', 'serpent']
```

J'ai utilisé le même contrat qu'avec geth, une transaction était bien générée (0xffc8450fd29e6dc26b7df84328913df16dad4b <https://etherchain.org/tx/0xffc8450fd29e6dc26b7df84328913df16dad4b29b3fbd1df9df2a5d12dabc mais apparemment jamais validée :

```
20:51:34|eth  New transaction ffc8450f...{[CREATE]/788$0+300000@50000000000<-0007ca35... #0}

'Contract transaction send: TransactionHash: 0xffc8450fd29e6dc26b7df84328913df16dad4b29b3fbd1df9df2a5d12dabc251
```

Cela illustre bien qu'Ethereum n'est pas encore prêt pour un usage généralisé. C'est une technique expérimentale mais très prometteuse techniquement et très intéressante politiquement.

Quelques lectures supplémentaires :

- Le Livre Blanc <https://github.com/ethereum/wiki/wiki/White-Paper>, la meilleure introduction technique à Ethereum.
- L'article de Gavin Wood <http://gavwood.com/paper.pdf>, un des concepteurs d'Ethereum, si vous voulez vraiment plonger dans les détails techniques. Aspirine de compétition nécessaire.
- Un dépôt de tas de contrats tout faits <http://ether.fund/contracts/>, pour les réutiliser ou bien pour apprendre.
- Un texte plus politique, pour un vaste public, de Vinay Gupta <https://docs.google.com/document/d/1ndBAcD72sHJnH0GpWY0pe8_KhkSZHFNTqBAHGBmY9N8/mobilebasic?pli=1>.
- Le Wiki de geth <https://github.com/ethereum/go-ethereum/wiki/> contient plein de bonnes informations.