Abstract

Ethereum is a relatively new blockchain based system which has become increasingly popular due to its innovative and unique nature. In this report an overview of what is Ethereum, why was it created and how does it work, will be given. In addition features like Smart contracts and Dapps are discussed. Ultimately we propose potential use cases for Ethereum platform and demonstrate one of them through a real application running on a local Ethereum network.
What is Ethereum?

Ethereum is an open-source blockchain-based distributed computing platform which provides a Turing-complete virtual machine (Ethereum Virtual Machine) for executing custom scripts on the network of public nodes. Ethereum also provides a currency called ether (ETH) which can be transferred between participants and is also used to compensate nodes for executing the scripts on the network.

History

Motivation

The first description of Ethereum was mentioned in a white paper written by Vitalik Buterin in late 2013 with the goal of providing a decentralized system capable of running entire applications with limitless capabilities. Buterin was an active Bitcoin network user so he compares Ethereum mainly with Bitcoin. Even Bitcoin has a weak scripting language built in however with several important limitations:

- **Lack of Turing-completeness** - for example there no possibility to write loops in Bitcoin scripts.
- **Value-blindness** - there is no way for a script to get any external information.
- **Lack of state** - scripts can’t have their own internal persistent “memory”
- **Blockchain-blindness** - since blockchain data is a potential source for randomness, such limitation severely limits applications in gambling and several other categories.

Ethereum has a way to solve each of these limitations. The project was funded by an online public crowdsale and developed by a Swiss company, Ethereum Switzerland GmbH in early 2014. The first live blockchain was launched on 30 July 2015.

Updates

From its launch Ethereum has had several updates. These updates are called **hard forks** since they are not backwards compatible and using the old version would fork the
blockchain from which only one path is considered valid. Generally these changes have seen positive response due to updates in stability and DDoS protection. However in 20 July 2016 a controversial decision was made to revert the state of the blockchain. A major bug in one of the applications running on Ethereum caused a potential loss of 50 million dollars. Such intervention was directly undermining the whole objective of a decentralized network and it's immutable applications (“Code is Law”) so some people decided to still use the old blockchain - a new currency was born: Ethereum Classic (ETC). Both of the forks are still actively being developed by entirely different people.

**How does it work?**

Ethereum can be viewed as a transaction-based state machine. We begin with a genesis state and incrementally execute transactions to reach the final (current) state. Valid transactions thus represent changes between successive states. Transactions are combined into blocks which are itself chained together using a cryptographic hash as the link between each other. Such chain of blocks forms the *blockchain*. New blocks can be generated into the chain by solving computational cryptographic problems — *mining*.

In order to incentivise solving the computation problems, there needs to be certain rewards. Ethereum has an intrinsic currency, Ether (ETH), which is rewarded to successfully solving the computational problems (finding a new block). Since all parties have the opportunity to create a new block on an older pre-existing block, there needs to be always a known “correct” path from genesis block to the leaf, otherwise a tree structure would be created. Such forks in the path are to be avoided at all costs. The agreed-upon scheme is to just always use the *heaviest* chain.

**Who is influencing the network?**

The world state is a mapping between addresses (160-bit identifiers) and *account* states. Such mapping does not exist directly in the blockchain but can be derived from existing data and stored in a local database.

Each *account* is composed of the following:
There are two types of accounts: **simple accounts**, also referred to as “non-contract” accounts which contain no executable code and **contract accounts** with executable code.

**What makes a block?**

Each block in the whole blockchain is composed of a block header $H$, set of transactions $T$ and a set of other block headers $U$ that are known to have a parent equal to the present block's parent's parent (ommers).

Each **transaction** is composed of the following:

- **nonce** - an integer, just a counter equal to the number of transactions send by the sender account
- **gasPrice** - value equal to number of Wei to be paid per unit of gas
- **gasLimit** - value equal to the maximum amount of gas that should be used in executing this transaction
- **to** - recipient 160-bit address
- **value** - value equal to number of Wei to be transferred to the recipient
- **v, r, s** - values corresponding to the signature of the transaction, also used to determine the sender of this transaction
- **init** (contract creation transcation only) - unlimited size byte array with the Smart contract's code

Each **block header** is composed of the following:

1. **parentHash** - Keccak 256-bit hash of the parent block's header
2. **ommersHash** - Keccak 256-bit hash of the ommers³ list portion of this block (blocks that are known to have a parent equal to the present block's parent's parent)

3. **beneficiary** - 160-bit address to which all the fees from successfully mining a block are collected

4. **stateRoot** - Keccak 256-bit hash of the root node of this state trie, after all transactions are executed and verified

5. **transactionsRoot** - Keccak 256-bit hash of the root node of the trie structure populated with all the transactions in this block

6. **receiptsRoot** - Keccak 256-bit hash of the root node of the trie structure populated with the recipients of each transaction in this block

7. **logsBloom** - Bloom filter composed from loggable information about information in the block

8. **difficulty** - value corresponding to the difficulty level of this block. It can be calculated from the previous block’s difficulty and timestamp

9. **number** - value equal to the number of ancestor blocks, initial block (genesis) has the value of zero

10. **gasLimit** - value equal to the current limit of gas expenditure per block (dynamic value depending on the activity in the network)

11. **gasUsed** - value equal to the total gas used in transactions in this block

12. **timestamp** - value equal to the output of Unix's *time()* function

13. **extraData** - arbitrary byte array containing data relevant to this block. The size of this data is limited to 32 bytes

14. **mixHash** - 256-bit hash which is combined with the nonce and can be used as a proof that a sufficient amount of computation has been carried out on this block (proof-of-work is valid)

15. **nonce** - 64-bit hash which is combined with **mixHash** and can be used as a proof-of-work verification

**Modified Merkle Patricia tree**

All the trie structures in the blockchain use Modified Merkle Patricia tree⁴ (hash tree) to store the data. Every node key value in the hash tree is calculated from their children hash values. This means that it is very easy to verify if the trie has been tampered with. Also the
trie is constructed in a way which provides $O(\log(n))$ efficiency for inserts, lookups and deletes.

**Gas and payment**

In order to avoid network abuse, every programmable computation in Ethereum is subject to fees. Such fee is specified in units of *gas*. Every transaction can limit its usage of gas by specifying a gasLimit. Initially the gasLimit is implicitly deducted from sender’s balance according to specified gasPrice. Any unused gas is returned to sender.

Miners are free to ignore any transactions they choose. This means a higher gasPrice on a transaction will deliver more value to the miner and thus more likely be selected for inclusion the block by the miners. This means there will be a trade-off between maximising the chance that their transaction will be mined and lowering the *gasPrice*.

**Mining Ether**

For a node to be allowed to add a block to the blockchain, it has to do a certain amount of work. The mining proof-of-work exists as a cryptographically secure nonce which guarantees that a certain amount $n$ of computational power has been spent. It also creates the possibility of adjusting $n$ - difficulty of mining another block. The distribution of PoW function should be as open as possible and would ideally make the act of mining a simple swap from electricity to Ether equally possible for everyone around the world.

The PoW function is designed to be sequential memory-hard, i.e. calculations are memory and bandwidth intense and can’t be parallelised. This is done to avoid creating specialized hardware only for the sole purpose of mining, as was demonstrated with Bitcoin ASIC miners\(^5\). The underlying proof-of-work algorithm used in Ethereum is called *Ethash*\(^6\).

**What is a Smart Contact?**

Smart contract can be thought as a program running in the blockchain. It is represented as an account with the additional executable code. Both the simple and contract accounts can interact with each other - send currencies and make code execution calls.
All data in the blockchain is public and immutable which means we are guaranteed that all the contracts behave exactly as their logic is written. Additionally no code can execute infinitely long since each operation is taxed.

**Code Execution**

All the code in Ethereum is compiled to a low-level, stack-based bytecode language, referred to as “Ethereum virtual machine code”. The most common languages that are used to write Smart contracts are *Solidity*, *Serpent* and *LLL*.8

Contracts can be added into the blockchain by any external agent. Since all the operations which manipulate and are run in the blockchain are taxed, generally the logic has to be as simplistic and optimized as possible.

**Decentralized applications (Dapp)**

Most web applications consist of a front and back end. The same idea is used in Ethereum decentralized applications where front-end is written in general web development languages as HTML, CSS and JavaScript but the back-end is running in the blockchain. Such applications are often called *Dapps*.

**Problems with Ethereum**

**Blockchain Scalability**

At some point the blockchain becomes too big for an average user to run a full node, thus full nodes become available only to a small collection of businesses that can have sufficient resources. If only a small number of entities store the entire blockchain, manipulating the chain becomes simpler. There are proposals of sharding the blockchain9 but it is a very tedious task and no good solutions exist to this day.

**Mining**

Proof of work as it is currently computed in the network is essentially a wasted effort. Miners spend 24 hours a day cranking out *Ethash* computations with the hopes of producing a block, and ultimately all of this work has no value to society. One of the
approaches to solving the mining problem is to abolish mining entirely and move to some other mechanism. The most popular alternative under discussion (and will be released in 2018) is **proof-of-stake** algorithm where the chances of person acquiring a valid block is directly proportional to amount of Ether on the account.

**Popular Dapps**

Since the EVM is Turing complete, there are unlimited possibilities for creating applications. Ideally the whole web would be running on Ethereum decentralised blockchain. Below, we list some of the popular applications.

- **Akasha** - social media application
- **EthLance** - job market
- **Golem** - market for computing power
- **FirstBlood** - gaming market
- **Ujo** - music provider (similar to Spotify)
- **RouleTH** - roulette
- **BlockJack** - blackjack

The list is even now much longer and probably it will be outdated shortly. As Ether itself is meant only for running and keeping the chain alive, there are already very many different tokens (currencies) build on top of the chain.

**Attachments**

With this report I've added a sample smart contract which simulates a simple gambling game. Once the contract is running in the chain, others can send money to the contract. With 50-50 probability the money will be either doubled and sent back or left in the contract. The original creator of the contract can cash out the Ether at any point.
References

2. https://github.com/ethereum/wiki/wiki/Mining
8. https://www.reddit.com/r/ethereum/comments/34gt50/contract_programming_language_solidity_serpent_or/?st=j34v2xq3&sh=f00ab3c9