# ETHEREUM ANALYTICS

Master Thesis

Athina Voulgari, Matriculation Nr.: 15-945-868
ETH Advisors: Prof. Dr. D. Sornette
UBS Advisors: Dr. V. Lange
ETH Zürich, Management Technology and Economics Department
Chair of Entrepreneurial Risks

05.06.2019

# Abstract

Ethereum is the second most valuable cryptocurrency, after bitcoin, with a market capitalization of $27 billion. Ethereum has attracted the interest of many cryptocurrencies fans, as well as investors, because it is not only a blockchain solution for the digital store of transactions' records, but also offers the possibility of creating and executing smart contracts.

Ethereum, and cryptocurrencies in general, have been in the center of criticism regarding their volatility and the speculation in their prices. In this thesis, in-depth analyses are performed regarding the Ethereum usability. The main focus is on understanding what the number of transactions in the Ethereum network represents and what the volume of ether in circulation between the different addresses is. For that purpose, the Google Big Query Ethereum database is used in order to extract all the available data and answer questions.

The number of transactions and the value of ether transferred are two good indicators to understand how the network behaves. Comparing these two, it is illustrated that they have not followed the same path and that they have not shown the same fluctuations during the last four years. The number of transactions has had an exponential growth, while the value of transferred ether has fluctuated a lot, on a daily basis. Moreover, the distribution of the number of transactions, as well as of the ether holdings among the different addresses, follow the 80%-20% pareto rule. More explicitly, 80% of the transactions are addressed to only 3.44% of the addresses and 80% of the transactions are sent from only 7.34% of the addresses, respectively. Regarding the ether holdings, 80% of the total ether is held by only 0.024% of the addresses. All these findings help us understand the speculatory nature of the Ethereum price. Finally, what's also exciting is the similarity of the smart contracts, as only 1.34% of the total contracts are completely unique.

Keywords: Ethereum, Blockchain, Smart Contracts, Ether Balance, Number of transactions, Contracts' similarity

# Acknowledgements

First and foremost, I would like to thank my professor Didier Sornette, who accepted me in his team and gave me the freedom to explore a research topic that I wanted to focus on, without adding barriers. I appreciate all his contribution and time he spent for my thesis.

Moreover, I would like to thank UBS AG, and especially my two supervisors, Dr Veronica Lange and Mani Mohanathas who, from their side, gave me the ground and time to explore a topic, not tightly related to the everyday business of UBS, and were open to discuss anytime.

When I started my master studies at ETH, it was an honour to receive a scholarship from Hellenic Petroleum SA. I owe a big thank you for this opportunity and for believing in me.

Furthermore, I really appreciated the time that some individuals spent with me, discussing about Blockchain concepts and adding one more piece in my knowledge and understanding path. A big thank you to Adrian Schmidmeister, Twan Sevriens, Ian Cusden, Evgeny Medvedev, and Sharat Koya.

A special thank you to Athanasios Stefanidis who was motivating me every day to continue, was the audience for all the concerns I had during this knowledge path and supported me, especially during finalizing the thesis.

Lastly, I would like to thank my family, my parents and my sister, for the unconditional love and trust they have shown to me during my whole educational path. They raised me with the value of trying hard to achieve whatever I would like to. I had them always in my mind during the hard times when I had to overpass difficulties and go forward.

# Contents

# List of Figures

# List of Tables

# Part I

# Main Text

# Chapter 1

# Introduction

## 1.1 Motivation

Our era is changing continuously. New disruptive technologies are being born constantly and totally change the way markets work. 2009 has been the year when Nakamoto introduced Bitcoin, a digital currency for peer to peer transactions, to the world. What he mainly achieved was not only to introduce the first digital currency but to open the doors for the whole cryptocurrency world. Crypto is not, anymore, a topic concerning only geeks and tech enthusiasts. As there is a total market capitalization of 333.86B as of $4^{th}$ of July 2019 [CoinMarketCap], any forward- thinking company and institution is not allowed to ignore the potential of blockchain market. Many other crypto initializations followed Bitcoin, but one has attracted the interest of many companies; that is Ethereum. Given the market cap ration of Bitcoin to Ethereum, it has been claimed that Ethereum has become more valuable than Bitcoin [D. Hao, 2018]. Whereas Bitcoin is a currency that operates only as a currency, Ethereum has a multifunctional character allowing users to create smart contracts that can interact and be executed. It does not serve only the transactional currency model. As a result, Ethereum offers the vision of a decentralized future, where third-parties are eliminated and users have direct access to the markets. Understanding how Ethereum works helps us identify what will be the blockchain use of tomorrow, how and when it makes sense to use it. Blockchain is not a panacea for all the innovation issues companies face. Despite its current trend, it cannot be the solution to everything and companies and institutions around the globe should first try to understand how the new technology can be effectively utilized before implementing a solution that can cost millions. The motivation of this thesis is to closely analyse the five-year route of Ethereum and explore some of its real use cases that generate added value.

## 1.2 Distributed Ledger Technology (DLT)

Distributed Ledger Technology (DLT) refers to a novel approach to recording and sharing data across multiple data stores (ledgers), which store the exact same data and are main-

tained and controlled by a distributed network of computer servers, which are called nodes [Natarajan et al., 2017]. The innovative feature is that the ledger is not maintained by a central authority.

Each node, independently, builds and records the updates in the ledger. After that, these updates must be accepted by most of the network. In order to overcome the challenge of ensuring that all the nodes agree upon the validity of the ledger updates, there have been different mechanisms that define the way of agreeing on this; this agreement is called consensus. Once the consensus has been reached, the distributed ledger updates itself and the latest, agreed-upon version of the ledger, is saved in each node separately [S. Ray, 2018a].

Regarding the consensus mechanisms, there are different ways of validating the agreement given the set-up of the distributed ledger. Consensus algorithms define the steps and the order of the steps that need to be done in order to produce an output. They constitute a fundamental component of distributed networks and are crucial for their functionality.

- **Proof of Work (PoW)**
  PoW existed before cryptocurrencies, but it became famous as it is used in Bitcoin and Ethereum. It would be very easy for someone to hack a trustless and distributed system where everyone can participate and verify transactions. However, PoW prevents an entity from gaining power over the whole network. For someone to verify a transaction in the network working with PoW, computational power is required. This transaction verification process is called mining. The purpose of mining is to verify the legitimacy of a transaction, avoiding any double spending. A group of transactions are first bundled into a memory pool. Miners verify each transaction separately by solving a complex mathematical puzzle. The first miner solving the puzzle receives as a reward coins of the respective network (bitcoins in Bitcoin, Wei in Ethereum, etc.). The new verified pool of transactions constitutes from now on a block and is attached to the blockchain as the latest block.
  Regarding the mathematical puzzle that the miners must solve, it is asymmetric making it difficult for miners to solve it but easy to be verified by the network [T. Schumann, 2018]. It does not require any mathematical skills, just computational power.
  This fact makes the PoW consensus expensive. It is estimated that the current energy power consumed for mining in Bitcoin equals the total energy need of Ireland. If Bitcoin transactions were to replace all the current transactions e.g. with Visa or Mastercard, the required electricity would exceed the current global consumption [Natarajan et al., 2017]. However, making block creation computationally "hard" prevents attackers from recreating the entire blockchain in their favour [V. Buterin]. Programming an attack to a PoW network is very expensive, and someone would need more money than he can be able to steal. After the production of several blocks, the mining algorithm dynamically adjusts its difficulty according to the hashrate [1] of the

[1] Measured in hashes per second.

whole network [Mingxiao et al., 2017].

- **Proof of Stake (PoS)**
  Proof of Stake is another way to achieve the agreement of transactions in the network. It is quite close to the PoW but the process is different. It was first introduced on a bitcointalk forum in 2011 [Blockgeeks, a]. In this consensus, the creator of a new block is selected in a deterministic way, depending on his wealth, also defined as stake. Miners need to put up some cryptocurrency as a collateral. This means that the validator must have assets in the respective currency. The more the stake, the greater the opportunity to mine a block successfully. It takes away the need for computational energy but brings the so-called nothing-at-stake problem [G. Konstantopoulos, 2017]. As the PoS consensus is not connected in the physical world, additional measures of security are required [Blockgeeks, a]. In case of a fork [2], validators could claim twice the amount of transactions and achieve double spending, as they will not have to split the computational power.

- **Delegated Proof of Stake (DPoS)**
  DPoS is an attempt to overpass the inefficiencies that both PoW and PoS deal with. Delegated proof of stake uses real-time voting combined with a social system of reputation to achieve consensus.
  In the network, there are delegates who are voted by token holders in order to mine the blocks [Shuwar and Vashchuk, 2018]. The number of based tokens an account is holding defines the voting weight (power) of each holder. It is very important that the delegates are selected in a rational way as they are responsible for the efficient and smooth operation of the network [Lisk.io]. In some networks, a similar-to-the-PoS concept of collateral is implemented in order to avoid misbehaviours.

- **Practical Byzantine Fault Tolerance (PBFT)**
  Practical Byzantine Fault Tolerance (PBFT) was initially introduced by Miguel Castro and Barbara Liskov in 1999. The algorithm has been designed in a way to detect malicious nodes in the network and defend against them. The consensus is reached by the fair nodes in the network. One node is named the primary one and all the rest constitute the backup nodes. All of them communicate and try to reach an agreement. The nodes need to prove that the message came from a specific source, as well as, that there have not been changes in its recording during its transmission [B. Curran, 2018]. Some of the advantages are the significant less energy usage and the ability of execute a transaction without the need of confirmation. However, this consensus can work only in consortium of participants and especially in a small-sized network. Moreover, it is vulnerable to attacks where a single validator can manipulate several nodes. It is mainly used in Hyperledger Fabric, which is extensively described in section 1.4.2.

[2] A fork is said to take place when a blockchain splits into two branches. Depending on the impact of the software change, it can be a hard or a soft fork.

Regarding the participation of users in a distributed network, there are two kinds of participation: permissionless and permissioned [M. Beedham, 2018]. As the name suggests, a permissionless DLT defines a network where no permission is required for the participation and the contribution to the network. Since anyone can join a permissionless DLT, these networks tend to be far more decentralized than the permissioned ones. The decentralized nature of the DLT means that the network does not rely on a central point in order to operate and be controlled. The lack of a single authority makes the system more transparent and secure. One example of a permissionless network, which can also be graphically illustrated in fig. 1.2, is the Ethereum.

Figure 1.1: Centralized Ledger [Natarajan et al., 2017]

On the other hand, if more control and privacy are required, permissioned DLT is a potential solution. The reason it is called permissioned is because access to the network requires permission. The owner of a permissioned DLT is responsible for managing who can participate and how a participant contributes to the network. Moreover, the owner of the network is responsible for any decision that needs to be taken into the network. In short, the owner or owners give or restrict access to the participants. Examples of a permissioned DLT, which is graphically illustrated in fig. 1.3, are Fabric and Corda.

Another distinction between the DLTs is the idea of a private or public DLT. It is important not to confuse the public/ private DLT distinction with the permissionless/ permissioned

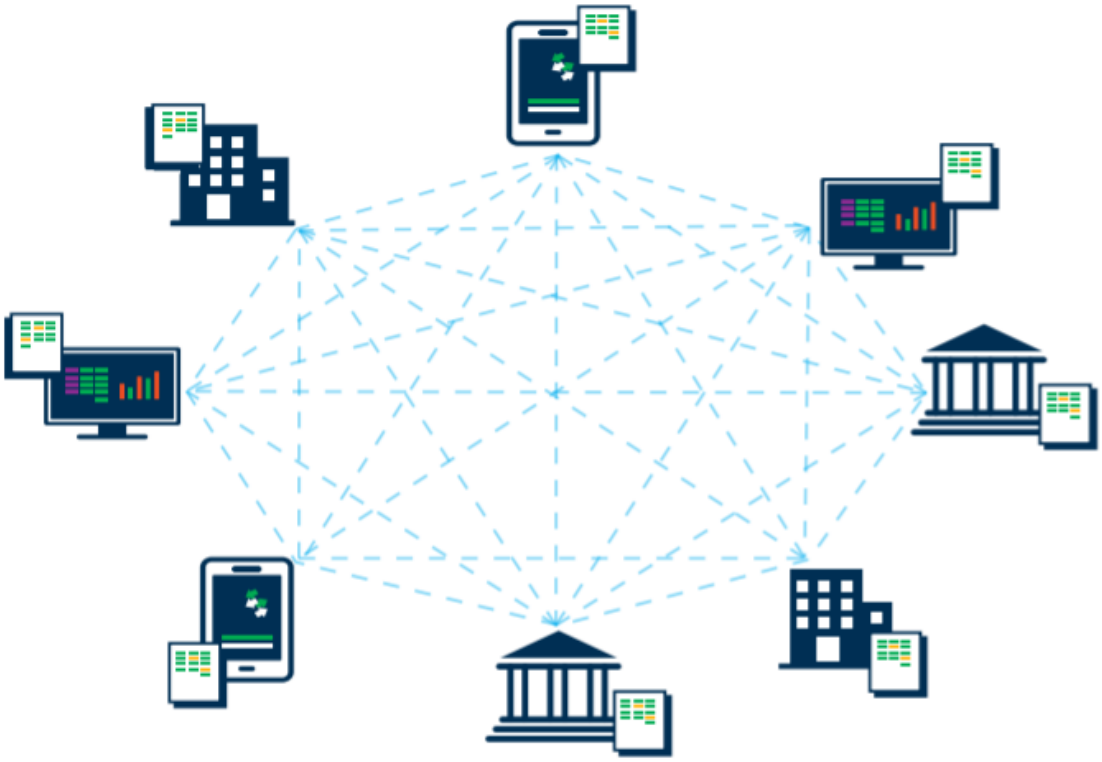

Figure 1.2: Distributed Ledger (permissionless) [Natarajan et al., 2017]

DLT distinction. The concept is that the public/private distinction has to do with the user authentication, while the permissioned/permissionless one refers the user authorization [Blocktonite, 2017]. The former checks who the user is and if the user has access to the network. The latter controls what kind of transactions the user is allowed to do in the network, and mainly refers to the right of validating transactions. The above is explained with a diagramm in fig. 1.4.

One basic difference between permissionless and permissioned DLTs is the speed of the network. Permissionless DLTs are often slower than the permissioned alternatives [M. Beedham, 2018], as they require each peer to execute each transaction and run consensus at the same time. As a result, this kind of consensus makes the network not scalable. Moreover, there are not confidential transactions as they can be verified by any participant in the network. However, permissioned DLTs reduce the ability to operate without the coordination of any single entity. Finally, the mode of participation (permissionless or permissioned) has a profound impact on the way consensus is reached [Valenta and Sandner, 2017].

As both network types serve different needs in the real world, there cannot be a binary categorization, but the precise features vary from platform to platform. There are many hybrid

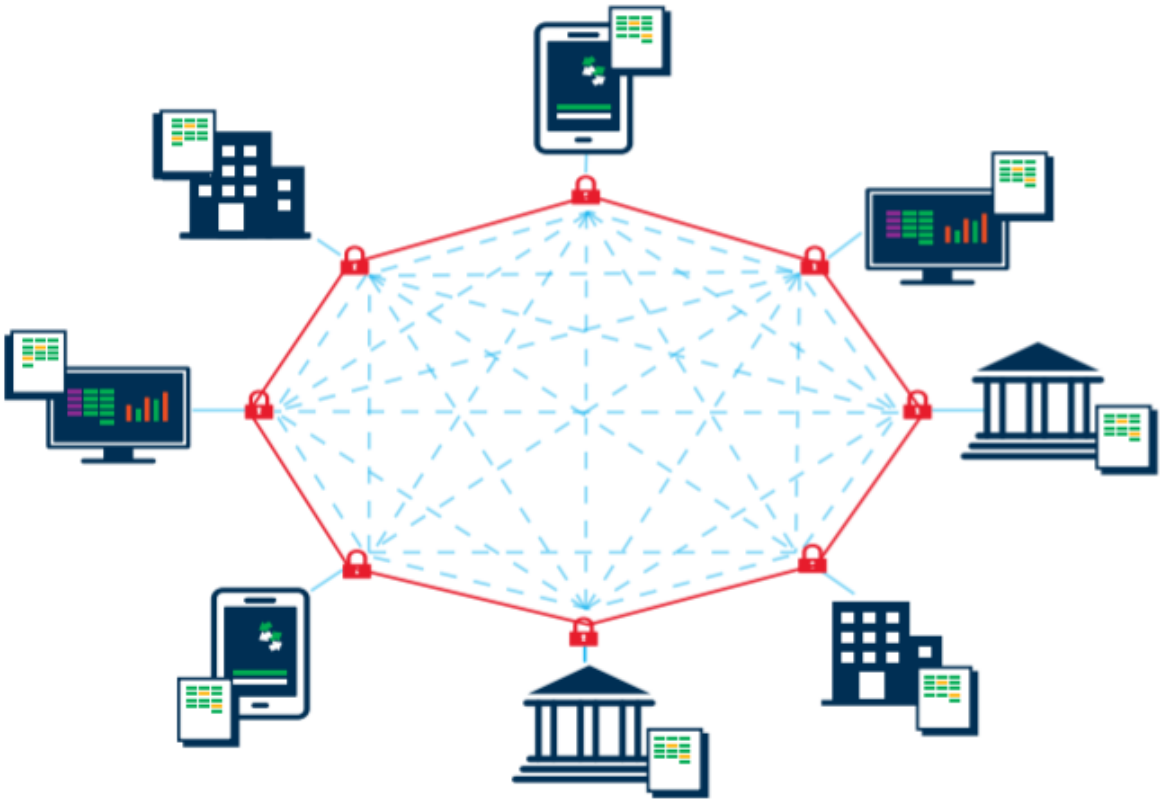

Figure 1.3: Distributed Ledger (permissioned) [Natarajan et al., 2017]

approaches trying to meet companies' needs for a permissioned network built on public blockchain infrastructure. An example is Enterprise Ethereum Alliance (EEA), where they try to build a private version of Ethereum's blockchain (see section 1.4.1) for its members to address their specific business needs.

DLT is the foundation of blockchain. According to H. Natarajan et. al., *A 'blockchain' is a particular type of data structure used in some distributed ledgers which stores and transmits data in packages called "blocks" that are connected to each other in a digital 'chain'* [Natarajan et al., 2017] (see section 1.3). However, blockchain is not the only DLT. Although blockchain is a chain of blocks, distributed ledgers do not necessarily require such a chain [O. Belin]. Alternatively, there are ingenious developments, such as Hashgraph and Directed Acyclic Graph (DAG).

- Hashgraph achieves transaction success solely via consensus through a gossip about gossip technique and a virtual voting technique. Interestingly, these techniques do not require proof of work to validate transactions [H. Anwar, 2018]. As a result, Hashgraph offers thousands of transactions per second while blockchain platforms, such as Bitcoin and Ethereum (see section 1.2), allow for approximately 5 transactions per second and 15 transactions per second, respectively [S. Ray, 2018b].

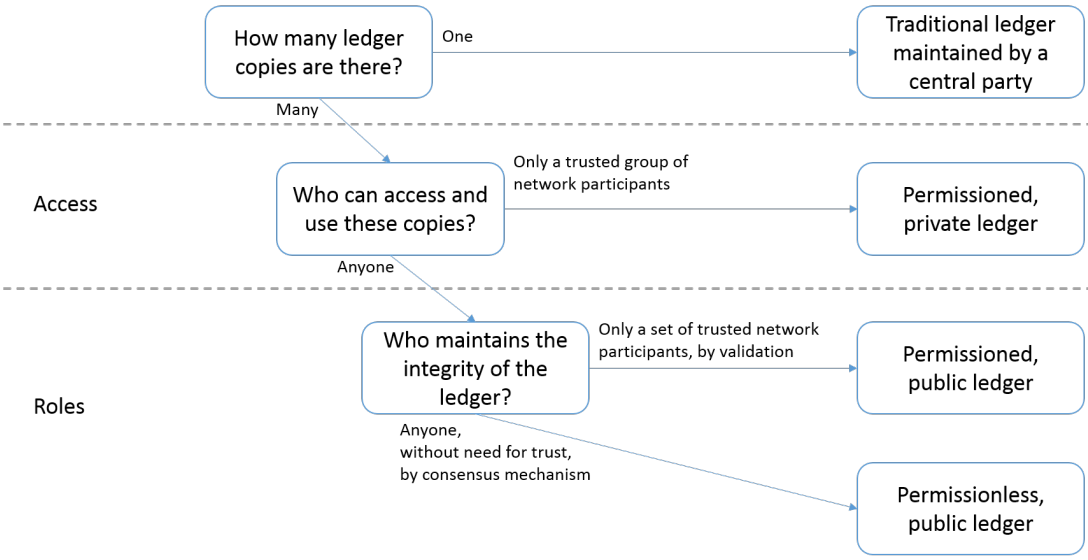


Figure 1.4: Distributed Ledger Taxonomy

- DAG is a type of distributed ledger technology that relies on consensus algorithms. These consensus algorithms operate in a way that transactions that prevail simply require majority support within the network. In such a network, there is much more cooperation and teamwork and nodes have equal rights [H. Anwar, 2018]. An example of a DAG DLT is IOTA's Tangle. According to M. Thake, before a new transaction can be validated in IOTA, it must first validate at least two previous transactions [M. Thake, 2018a]. These two transactions are being randomly selected by an IOTA algorithm, restraining the members of the network from choosing their own transactions.

## 1.3 Blockchain Technologies

Blockchain was the first fully functional Distributed Ledger Technology and the only one people have been knowing for many years [M. Thake, 2018b]. As a result, there is even today a confusion between DLT and blockchain, serving as a good example for a service or product that overtakes "the umbrella" it is part of and ends up becoming its namesake. However, blockchain is nothing more than a DLT with a specific set of features that distinguish blockchain from all the previously referred DLT technologies [BBVA, 2018]. Blockchain is a database, shared by means of blocks, which are connected with each other in the form of a chain.

Blockchain was mainly mentioned in the Bitcoin white paper from Satoshi Nakamoto, published in 2008, as the underlying technology of Bitcoin. However, blockchain was not built from scratch in order to serve the Bitcoin concept. The technologies that are utilised

are not new. What is new and disruptive is the combination of these technologies to create the world's first ever digital currency. Some of these technologies are the peer-to-peer network, cryptography, digital signatures, nodes, hashing, consensus protocols, and mining.

Blockchain offers the main functionalities of DLTs. The three main properties that determine the Blockchain technology are decentralization, transparency, and immutability. The term of transparency comes in contradiction to the concept of privacy that is claimed to be a feature of blockchain. A person's identity is represented by a public address which is cryptographed. It is not the name of the sender that is referred to in a transaction, but the hash representation [3], such as "1Mftz7sFLkBfjh9vpFYEmvwT2Tb6Ct7NZJ". As a result, the physical identity of a user is secure, while the digital identity is monitored and recorded, so that transparency is secure as well. Furthermore, blockchain ensures that no recorded transaction can be altered or removed. Even the slightest change in a block will completely change the hash address of the block (how adjacent blocks are interconnected will be analyzed further in this thesis). This immutability feature makes blockchain fully reliable.

Finally and as described earlier, in a decentralized system, the information is not stored in a single entity. On the contrary, a basic component of a blockchain is its peer to peer (P2P) network. Each user, which is considered to be a single node, can use the network and provide resources at the same time. A node can be any active electronic device that is connected to the Internet and provides an IP address. Although all nodes are equal, they can serve the network in different ways. A special reference needs to be made to these nodes that are called "full nodes" or else miners [Lisk.io]. The purpose of a full node is to copy the latest blockchain to a single device, while the device is connected to the network. As a result, this information is stored in all full nodes of the network and can only be altered if all these nodes are destroyed, which makes the system less vulnerable.

### 1.3.1 Mining

Previously, a reference was made to these nodes that are called miners. Mining empowers the decentralized blockchain. Mining is the process where miner nodes use computational power in order to verify transactions and add them in the public ledger. Each miner ensures to maintain the latest version of the ledger which contains all the blocks. Each node competes against another in order to solve a mathematical problem, which is called nonce. The node which manages to solve first the puzzle updates the blockchain by adding the latest block. This is the reason, as well, that a lot of computational power is required. In order to incentivize miners to do their job, a reward of the respective currency of the network is given to the miners. This approach of validation follows the proof of work consensus to which we have refered earlier (Ethereum and Bitcoin have different kinds of reward).

[3] Hashing is the process of taking an input of any length and turning it into a cryptographic fixed output through a mathematical algorithm (Bitcoin uses SHA-256, for example) [Lisk.io]

### 1.3.2 Digital keys, cryptocurrency addresses and digital signatures

Having talked about the hash representation of users and about the transparency, combined with privacy, it is important to mention the way users are represented in the network and explain how they are capable to participate in transactions privately. Cryptography is a basic component. Although diving deep into cryptography is not a desired outcome of this thesis, some basic elements, which make use of cryptography, in a blockchain, such as digital keys, cryptocurrency addresses and digital signatures need to be explained.

Digital keys are used from users in order to perform transactions. They are generated in pairs; for every public key there is a also private one. The public key is the identity of an account to the network and is visible to anyone, while the private key is the "password" of a user account and is only known to the user in order to unlock the respective account. Both keys and the account are not stored directly in the network, but in wallets which exist independently from the network. The hashed address is the representation of the public key in the network and is called cryptocurrency address [B. Asolo, 2019]. On the other side, the private key is used to sign transactions and is referred to as the digital signature, meaning that it gives a user access and control over the funds.

### 1.3.3 Conclusion

*Blockchains have the potential to build a new generation of transactional applications that establish trust, accountability, and transparency at their core while streamlining business processes and legal constraints* [Blockgeeks, b]. They also enable the maintenance of a historical record of all transactions, as well as the means to add and administer new entries.

## 1.4 Comparison of some featured Blockchain platforms

### 1.4.1 Ethereum

Ethereum was introduced in Vitalik Buterin's paper and its innovation is that it supports all types of computations. It is an open-source, blockchain-based platform, based on rewards, which eliminate the need for trusted intermediaries. According to Dr. Gavin Wood, co-founder of Ethereum organization, Ethereum, taken as a whole, can be viewed as a transaction-based state machine: we begin with a genesis state and incrementally execute transactions to morph it into some final state. It is this final state which we accept as the canonical "version" of the world of Ethereum [G. Wood]. It allows everyone to set their own rules in transactions and state transition functions. This is done by involving smart contracts, a set of cryptographic rules that are executed under the conditions that the creator of the contract has set up.

Ethereum is a novel cryptocurrency that uses a blockchain not only to store a record

of transactions, but also to store user-generated programs called smart contracts and a history of calls made to those contracts. A blockchain with a built-in Turing-complete programming language, allowing anyone to write smart contracts and decentralized applications where they can create their own arbitrary rules for ownership, transaction formats and state transition functions [V. Buterin]. Instead of using the computer system of a big company like Google (a centralized system), Ethereum lets software applications run on a network of many private computers (a decentralized system) [Upfolio].

Ethereum allows developers to program their own smart contracts, or "autonomous agents", as the Ethereum white paper calls them. The language is "Turing-complete", which means that it supports a broader set of computational instructions. In a nutschell, smart contracts can:

- Function as "multi-signature" accounts, so that funds are spent only when a required percentage of people agree
- Manage agreements between users, e.g. if one purchases an insurance product from another
- Provide utility to other contracts (similar to how a software library works)
- Store information about an application, such as domain registration information or membership records

One trade-off of Ethereum is related to its consensus mechanism (Proof of Work). It is the most popular algorithm for verifying the correctness of a transaction happening in the network. On the one hand, it requires a lot of computational power in order to verify any transaction and at the end to build a block and validate it. On the other hand, the fact that it requires so much computational power makes it difficult to replicate and change the whole chain and as a result, reduces the risk of a 51% attack. At the same time, each individual solution is easy for the community to verify, which makes it easy to check all transactions for trustworthiness. The fact that it doesn't rely on a single third-party entity makes it a "trustless" and transparent network. PoW also sets a limit on how many new blocks of data can be generated. For example, miners can only create an Ethereum block every 10-20 seconds.

### 1.4.2 Hyperledger Fabric

Hyperledger is an open source community of communities that benefit from an ecosystem of Hyperledger based solution providers and users, focusing on blockchain-related use cases that work across a variety of industrial sectors.

Fabric is one of the projects of Hyperledger, under the auspices of the Linux Foundation. Fabric is used in many prototypes, proofs-of-concept, and in production distributed ledger systems, across different industries and use cases. It is about a consortium blockchain network in which organizations can participate as "members" given that they have the permission to participate. Each member is responsible for setting up the peers that are required in order to participate in the network. Moreover, it can create its own network which results in a scale up to more than 1000 transactions per second [Blockgeeks, b].

Fabric has no native currency and as a result, the definition of what can be transferred is wide and can be defined as asset. An asset can be anything with monetary value. Assets are represented as a collection of key/value pairs, with state changes being recorded as transactions on the ledger. Members can interact with the ledger by using the chaincode which defines the rules, the state and the business logic of an asset. This chaincode gives the possibility to build a private channel for asset transactions. The chaincode needed to read and alter the state of an asset will only be installed on peers involved in this certain business case, which means that Fabric's blockchains allow the users to participate in private interactions [M. Beedham, 2018].

As a result of the permissioned nature of Fabric, there is a membership identity service that manages IDs, authenticates participants, and provides additional layers of permission. Only parties directly affiliated with the deal are updated on the ledger and notified. This helps maintain privacy and confidentiality.

The consensus with which Fabric works is Practical Byzantine Fault Tolerance (PBFT), which checks how much time a transaction has resided on a machine. The main programming languages are Golang and Java. There are two kind of nodes: Peer nodes that execute and validate transactions and Ordering nodes that order and propagate transactions. It constitutes of two main parts: Blockchain logs that store the immutable sequenced record of transactions in blocks and the State database that maintains the blockchain's current state.

Summarizing the main functionalities of Fabric:

1. Users can define asset types and the consensus protocol
2. Users can set permissions on who can join the network
3. There are 2 distinct roles: users and validators
4. It consists of a log of transactions and a database of the current state
5. Assets are added, updated, and transferred by chaincode

Fabric does not focus on cryptocurrencies and monetary assets as most of the blockchains do, but in industrial applications of the blockchain technology. It provides a more fine-

grained access control to records and thus enhances privacy. Furthermore, a gain in performance is achieved as only parties taking part in a transaction must reach consensus [Valenta and Sandner, 2017].

### 1.4.3 R3 Corda

Corda is an initiative of R3, a company which builds distributed ledger technology and was founded in 2014 by David Rutter in New York City. It started as a consortium with nine financial companies: Barclays, BBVA, Commonwealth Bank of Australia, Credit Suisse, Goldman Sachs, J.P. Morgan, Royal Bank of Scotland, State Street, and UBS [J. Kelly, 2015]. Later, more financial companies started joining the consortium. As a result, Corda focuses on providing distributed solutions mainly in the finance industry.

According to the respective white paper, Corda is a distributed ledger platform for recording and processing financial agreements [R. G. Brown and J. Carlyle and I. Grigg and M. Hearn]. It is mostly designed as DLT for financial services because of the highly regulated environment it takes into consideration. That doesn't mean, however, that it is limited in blockchain concepts in other industries.

Corda is a permissioned, private network where all participants have been permissioned to participate, and their identity is verified. This permission is obtained in the form of a certificate which is received from the network operator who defines the rules for participation. Unlike other DLTs, Corda does not use the blockchain logic to record transactions. The reason is that Corda wants to offer the appropriate privacy to its participants. As a result, the parties who are involved in a transaction are the only ones who are informed about the transaction. This concept differs from the blockchain ledger sharing approach that Ethereum and Fabric use. In the common blockchain concept, networks broadcast all the transactions which are distributed to all the nodes of the network.

In the permissionless networks, where there is not trust among the participants, consensus needs to be reached by computational effort. In permissioned networks, the agreements are reached thanks to the participants' trust to each other. Corda goes a step further and instead of communicating the message to all the trusted participants, uses direct communication only to the involved parties, which makes the process much quicker and creates a veil of privacy, a key element in the financial industry. Hence, there is no single central store of data and each node maintains a separate database of known facts. In other words, each participant can only see a part of the ledger, the part in which he is involved [corda.net].

Furthermore, the basic component of Corda is the state. It is an immutable object representing a specific agreement or contract and it may contain any kind of information. The difference with the contracts that we meet in other blockchains is that the state is specific for a pair of participants and not an element used by the entire ledger. A state cannot be modified and in order to update it, a new version of the state is created.

Another important element of Corda are its transactions or flows. They represent the intention to update the ledger. A transaction can be accepted only if it does not contain double spending, is valid, and is signed by the involved parties [corda.net]. Transactions represent a link among the different states.

Finally, there is the notary which provides the definiteness in the system. It is the entity that provides transaction ordering and timestamp services [N. Avramov, 2019].

Overall, it can be claimed that Corda, although it is commonly considered as a blockchain, is not an actual blockchain, since it is not organised into blocks. However, it is a game player in the future of blockchain technologies and generally of distributed ledgers.

# Chapter 2

# The Ethereum Model

## 2.1 Accounts

Accounts play an important role in Ethereum. There are two kinds of accounts: Externally Owned Accounts (EOAs) and contact accounts, known as smart contracts as well. All the accounts in blockchain are characterized by four core elements:

- The nonce, a counter that is used to ensure that each transaction can only be processed once representing the number of transactions successfully sent from this account (if it is an EOA), or the number of contracts created by it (if it is a contract account).
- The account's current ether balance (the number of wei owned by an account).
- The account's code (only for contract accounts).
- The account's storage (permanent data stored, only for contract accounts) [V. Buterin].

### 2.1.1 External Owned Accounts (EOAs)

EOAs are crucial for users in order to interact with the Ethereum blockchain by using transactions and they represent the identities of external entities. An EOA is controlled by the pair of keys that are mentioned earlier in this thesis, the private key and the public key. Accounts are indexed by their address, which is the last 20 bytes of the public key. The private key, which is a 20-byte address as well, is always encrypted with the password the user selects when the accounts is created. The public and the private key are stored in a keyfile.

For the time being, it is very important for any user to carefully store the keyfile. If the password for the keyfile is lost, there is no way to retrieve it, which means that any amount of ether in the account would become inaccessible and "lost" in the network. In

order for a user to create such an account, the non-coding way, is to use the official Mist Ethereum wallet, which is developed under the Ethereum Foundation.

External accounts can initiate transactions either by transferring ether or by triggering some contract code. Moreover, as seen earlier, an external account might have a balance, but they maintain neither bytecode nor storage, in contrast to the smart contracts. This difference will help the upcoming analysis, where the two types of contracts will be compared. Finally, transactions in the Ethereum network can only be initialized from EOAs.

### 2.1.2 Smart Contracts

A smart contract, or contract account, is another core element of the Ethereum network and the key differentiator of Ethereum from other altcoins[1]. Smart contracts constitute one of the most successful applications of the blockchain technology. The reason is that Ethereum supports the feature, called Turing-completeness, that allows the creation of customized smart contracts.

Contracts exist in the network as independent entities that execute a part of their stored code when they are getting triggered by either transactions or messages. This stored code is called bytecode. The bytecode stores all the rules of an agreement between different parties, validates and executes the agreed terms automatically, without relying on third parties. It consists of different functions. These functions define the business logic, which is executed once the contract has been deployed. In order to save space and keep the database clean, Ethereum allows contracts to be self-destroyed when they do not add value anymore. To incentivize the developers to create contracts with this functionality, Ethereum refunds the amount of ether that has been spent to create the contract.

A smart contract is represented by its 20-byte address, similar to an EOA address, such as '0xd1ceeeeee83f8bcf3bedad437202b6154e9f5405', and it can be developed in high-level languages, such as Solidity[2]. When the contract is deployed on the network, the solidity code is compiled as bytecode.

The smart contract functionality is the key element that makes Ethereum so flexible for different applications. Ethereum allows developers to program any smart contract or differently "autonomous agent", according to the white paper [V. Buterin]. As a result, smart contracts need to have the following three characteristics:

- Deterministic: Given the same inputs in different computers, the output will always remain the same.

[1] "Altcoin" is a combination of two words: "alt" and "coin"; alt signifying alternative and coin signifying (in essence) cryptocurrency. Most of these altcoins are built up on Blockchain. However, they differ themselves with a variety of different attributes, such as different consensus and functionalities

[2] Solidity is an object-oriented, high-level language for implementing smart contracts [Solidity, 2019]

- Terminable: A contract has to be able to terminate after a given time limit. In order to achieve this, contracts use steps counter or timer and terminate when the execution exceeds the defined limit.

- Isolated: The contracts are kept isolated in a sandbox. As a result, it is secured that the entire network is not under risk.

However, the flexibility to create any contract can sometimes be seen as a drawback and together with the complexity of Solidity, these are the main reasons why there are many contracts in Ethereum that are vulnerable to attacks, as they have not been designed taking into consideration all the risks.

**Lifecycle of a smart contract**
The initial phase of a smart contract is the creation. This phase includes the definition of the contract objective and the contract rules (what the contract is supposed to deliver) and the coding implementation. After transforming the contract requirements into contract code, the coded version of the contract is submitted to Ethereum. After the code of the smart contract is created, it cannot be modified. If anything has to change, a new contract is required to be created. There is a difference between the code used when a user creates a contract in the Ethereum platform and the code of the contract itself. In order to create a contract, a transaction needs to be executed towards a special "0x0..." address and an initiation code needs to be added as data in the respective field. This initiation code in not the same as the code of the new contract. The initiation code, called contract deployment code, is used by the EVM in order to initiate the contract, while the output of this execution is stored as the code of the contract [Sillaber and Waltl, 2017].

When a contract is submitted to the blockchain, it is getting exposed to miner nodes in order to confirm its validation. In order to validate the new contract, a fee (gas) has to be paid to the miners as they spend an amount of computational power. From this point on, the smart contact is public and accessible by all users in the network.

In order for users to use a contract, they need to send some input for execution. When the contract receives this input, the respective functions are executed. As a result, this execution creates a set of new transactions and a new state of the contract. All these new elements are submitted to the Ethereum ledger and are validated through the consensus mechanism [Sillaber and Waltl, 2017].

Finally, the respective committed digital assets are transferred to the appropriate party and the contact has been fulfilled.

There is an infinite amount of contracts that can be created in the network. However, there are some "templates" that are commonly used, as they provide some main functionalities for contracts that want to achieve similar behaviours. These are token standards

that define a common list of rules and are called ERC (Ethereum Request for Comments). Some of these are described below.
**ERC20:** This standard is a simple interface for creating tokens and can be reused from any application. It is the most commonly used standard in Ethereum. It consists of 6 main functions and 2 events. These functions are:

- Total Supply: defines the initial total supply of tokens
- Balance: monitors the balance of the contract every time
- Transfer to: sends tokens to specific accounts (wallets)
- Transfer from: enables token holders to exchange tokens after the initial distribution of the tokens
- Approve: approves other accounts to withdraw tokens from the account calling the function
- Allowance: after the approval of withdrawing tokens, it is used to see the amount of tokens that is withdrawn from the account

ERC20 is used from wallets in decentralized exchange platforms. Moreover, it is the main standard used for ICO contracts [V. Lai, 2018].

**ERC223:** If ERC20 tokens are sent to a smart contract that it is not built on an ERC20 standard, these tokens are not accessible anymore. For this reason, ERC223 supports an additional functionality that ensures that tokens are only sent to contracts with the right standard.

**ERC721:** This is a standard developed for non-fungible[3] tokens. These tokens are completely unique. This feature of the standard makes it suitable for representation of assets that cannot be duplicated. A good example is the ownership over assets, such as houses, art pieces, or digital collectibles.

As mentioned earlier, smart contracts are simply some lines of code that define a set of rules and are executed when the smart contract code is activated. In order to execute a contract, a user uses a **Decentralized Application** (DAPP). DAPPs are software applications that are connected and operate in the Blockchain through smart contracts. This is the way that users can interact with smart contracts. They can be seen as an interface of a smart contract running on the Ethereum network.

[3]A fungible token is a token that can be replaced by something identical and it is interchangeable such as an ETH token. A non- fungible token contains unique information that make the token irreplaceable.

## 2.2 Transactions

As seen earlier, Ethereum is an account-based model, as our known banking system. The state of Ethereum changes every time a transaction of value or information is executed between two accounts. So, the state[4] consists of these account addresses.

There are three kinds of transactions:

- **Ether transfer:** An external account can transfer Ether to another external account or a contract. This kind of transaction is very similar to a Bitcoin transfer.
- **Contract creation:** An external account can create a contract by transferring ether to a zero recipient's account. Then a new contract is created. The transaction should contain a code defining what the new contract will do. The execution of the transaction will create the bytecode of the new contract.
- **Contract call:** After a contract is deployed, an external account can call a contract when an account intends to implement one or more functions of a contract. The input data contain all the instructions related the execution of the contract.

In order to execute all the different transactions, it is very important to set correctly all the different parameters, which will be analyzed up to a broader extent later.

Smart contracts cannot initiate transactions. However, a contract can send an internal transaction to a another contract account. This internal transaction is called a message call that is part of the initial contract account's code and is indirectly fired from a transaction that is originally initialized by an external account. Thus, the message calls can be seen as an extra type of transaction that takes place between contract accounts. In case of such a message call, this message contains the functions that should be activated in the called contract.

## 2.3 Ethereum Virtual Machine

Ethereum Virtual Machine (EVM) is the computation engine of the Ethereum network and handles smart contracts deployment and execution. EVM is required in order to update the Ethereum state. An ether transfer from an account to another does not require an update of the state and as a result, EVM is not involved. However, it is used for any other action in the network.

Specifically, EVM is responsible for all of the following executions in the network:

[4]It is part of Ethereum's base layer protocol. According to Ethereum yellow paper [G. Wood], Ethereum is a transaction-based "state" machine; a technology on which all transaction-based state machine concepts may be built.

- EVM confirms the validity of a transaction. This means that it checks the correct number of values, the validity of the signature and the match of the nonce with the nonce of the particular transaction account. If anything of these elements is not correct, an error will return back [Katalyse.io, 2018].
- EVM checks if the gas is enough in order to execute the transaction. In case gas is not enough, the transaction fails. However, the transaction fee in such a case would not be refunded and would be paid to the miner.
- EVM transfers the value of Ether to the recipient's address.
- EVM calculates the total gas used and the transaction fee, in order to initialize the gas payment to the miner.

## 2.4 Ether and Gas

Ether is the currency of Ethereum. It has intrinsic value as any currency. There is a misunderstanding between ether and gas. Gas is a commodity, like crude oil. It is the fuel for operating Ethereum. Every time a transaction or a contract is executed, users spend tokens which are translated to gas to run the computations. A user has to pay for the computational power spent, regardless of whether the transaction was successful or not. Gas is therefore an implicit incentive for developers to produce contracts that are of low-cost execution and for miners to validate transactions. Like this, the community avoids a waste of resources and eliminates the motivation for denial-of-service attacks.

Ether derives its value from different factors. First, as mentioned earlier, it is used to pay transaction fees in Ethereum. Furthermore, it can be accepted as payment currency from some retailers, such as the largest online Swiss retailer Digitec Galaxus AG. It can also be lent or borrowed. Finally, it is used as a medium of exchange to purchase Ethereum-based tokens. One example of the latter is an Initial Coin Offering (ICO), which will be analyzed later [A. Sassano, 2019].

Hence, ether serves all the three conditions in order to be considered as money within an economy.

- It works as a store of value, where investors purchase it and hold it for investment purposes given its predictable supply growth and congenital utility.
- It serves as a medium of exchange within the Ethereum network, and not only.
- It is used as a unit of account by various parties (including companies that have raised Ether via ICOs) [A. Sassano, 2019].

The smallest denomination of Ether's metric system is Wei, where 1 Ether = $1e^{18}$ Wei.

## 2.5 Blocks

Even by the name of the technology, it can be easily derived that blocks are a fundamental component of Ethereum. All the information is stored in blocks. Each block consists of a block header and a set of transactions mined in the block. The chain is created by hashing some of the information of the previous block and including it in the new one [ETH events, 2018]. The hash IDs and timestamp associated with each block are the virtual glue that connects the blocks and forms the blockchain. Any alteration of the ledger or an individual block will make the code unworkable and, as a result, would be detectable. Until now, no blockchain has been breached or hacked, although several cryptocurrency platforms have suffered breaches, reinforcing the importance of robust password security and protocols [Appelbaum and Smith, 2018].

The Ethereum blockchain begins its life with its own genesis block. After the zero-block initiation, all other activities, such as mining, transactions, and contracts change every time the state.

In Ethereum, the time between blocks is around 14 seconds, which means that Ethereum updates its state every 14 seconds. Each block is a pool of transactions that are stored together with the hash of the previous block and the proof-of-work of the current block [F. Gadaleta, 2018]. As a result, the proof-of-work and the content of the block constitute the hash of the new block. Currently the maximum block size in Ethereum (or else the block with the most computational power needed) is around 1'500'000 Gas.

## 2.6 Mining

Mining is the process of validating new transactions and creating new blocks. As a result, miner nodes will collect transactions from the transaction pool, will execute them in the EVM, solve the nonce and create the new block that fits to the chain. In Ethereum, the miner of a block receives rewards from 3 different sources:

- Three Ether for every block that the miner adds in the chain.
- All the gas the miner spends in order to validate the transaction of the block. This is in fact the compensation for the computational power spent.
- An extra reward for including Uncles[5] as part of the block, in the form of an extra 1/32 per Uncle included.

However, the whole way of validating and receiving rewards would change in a proof of stake consensus and would reform the way mining is done in the Ethereum blockchain.

[5]Uncle blocks are the ones which are created form miners but they are not included in the chain as the latest one as they were not the first one produced.

## 2.7 Data structuring and format

Ethereum has a very special way of handling data and processing transactions. The core data structure in Ethereum consists of Merkle Patricia trees (tries), a combination of Merkle and Patricia trees. A simple Merkle tree is a tree of hashes. Parent nodes contain the signature of the data of the child nodes as a hashed value of the sum of their childrens' hashes [K. Kim, 2018] (see fig. 2.1). On the other side, a Patricia tree is a data structure which is also called Prefix tree, radix tree or trie. A trie uses a key as a path, so that the nodes that share the same prefix can also share the same path [K. Kim, 2018] (see fig. 2.2). The Merkle tree guarantees the efficient and secure verification of the data through this hashing process, while the Patricia tree allows an easy search through the nodes.

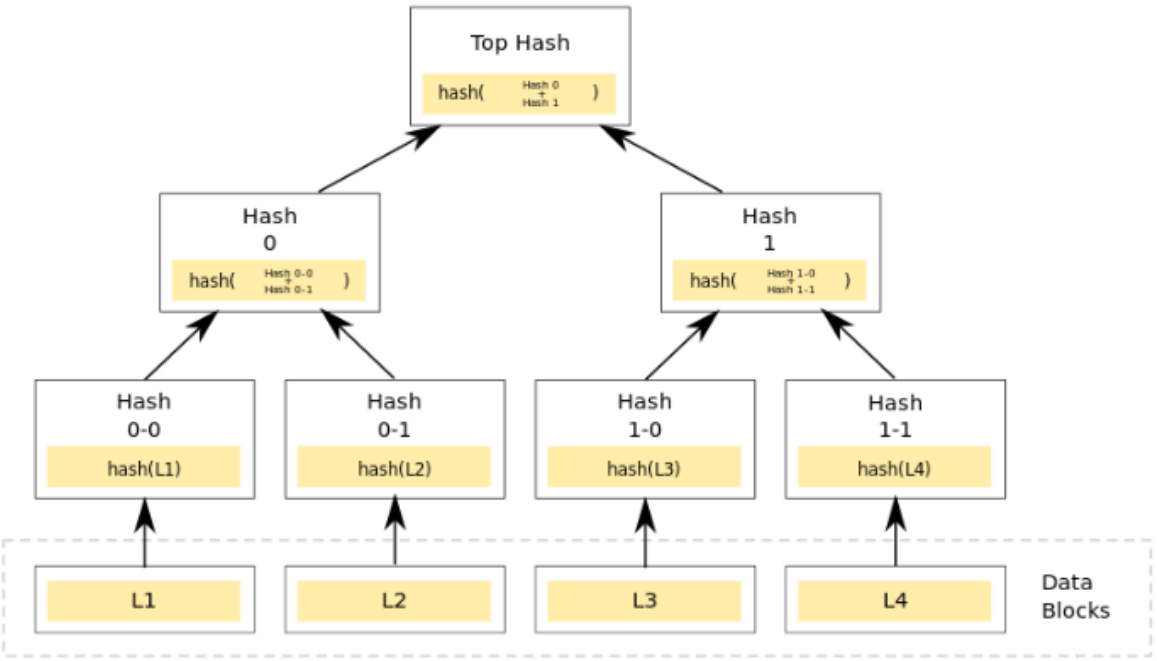


Figure 2.1: Example of Merkle Tree [K. Kim, 2018]

A Merkle Patricia tree is, therefore, the combination of both in order to increase the performance and the efficiency, providing a persistent data structure to map data. Similarly to the Merkle tree, every node has a hash value again, which is the key identifier of the node. The Patricia structure comes to add a node path where nodes that share the same prefix can also share the same path. This structure is fastest at finding common prefixes, simple to implement, and imposes small memory requirements [K. Kim, 2018].

Ethereum uses this data structure for each of the four data tries it maintains: receipt

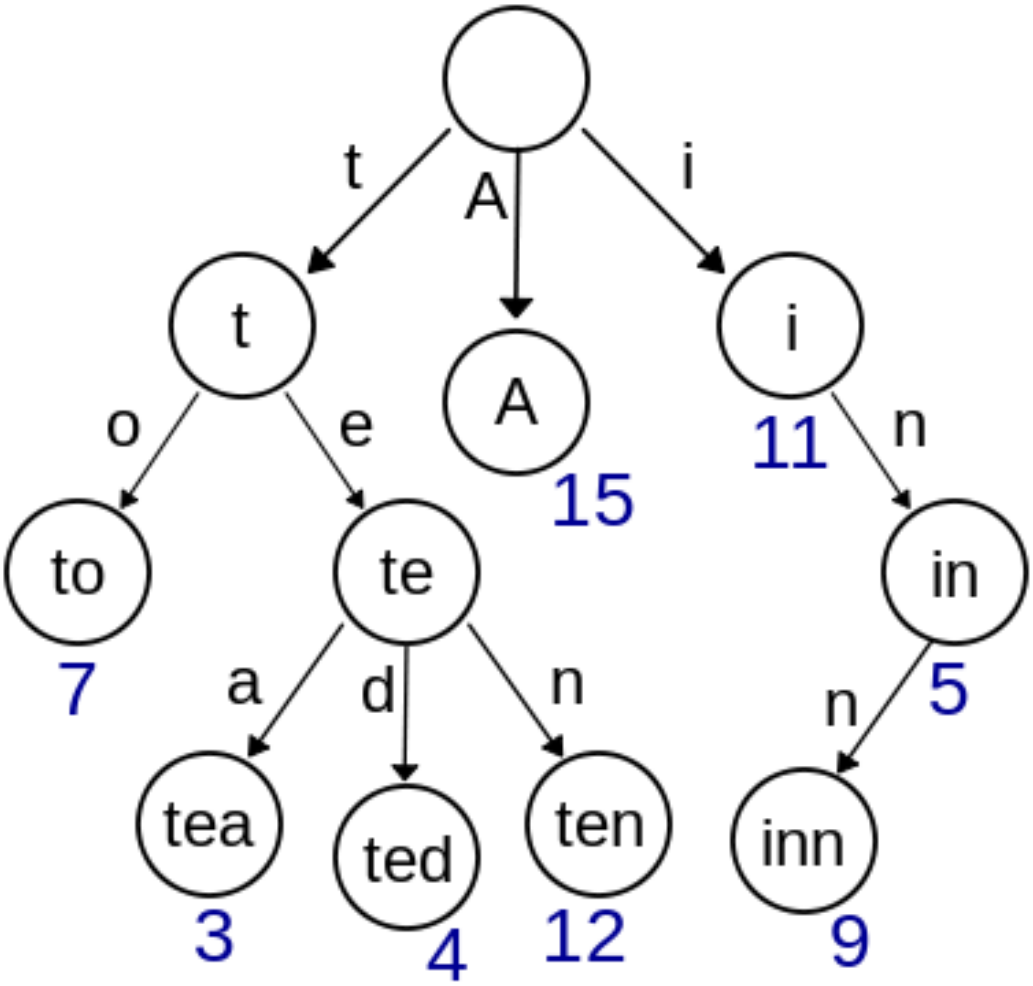


Figure 2.2: Example of Patricia Trie [K. Kim, 2018]

trie, state trie, storage trie, and transaction trie. Receipt, state and transactions tries constitute the block header of each separate block. In detail, only the root node hashes of the transaction trie, state trie and receipts trie are stored directly in the blockchain. The storage trie's root is part of the state trie at the very end. This is illustrated in fig. 2.3.
The way the tries are working is out of the scope of this thesis. However, it is important to understand that these four tries (state, transactions, receipts, and storage) contain all the data of a block.

- State trie provides a mapping between addresses and account states and maintains the global state. It is unique in the Ethereum network and is constantly updated. It consists of the nonce, balance, storage root, and the codehash [D. Brickwood, 2018].

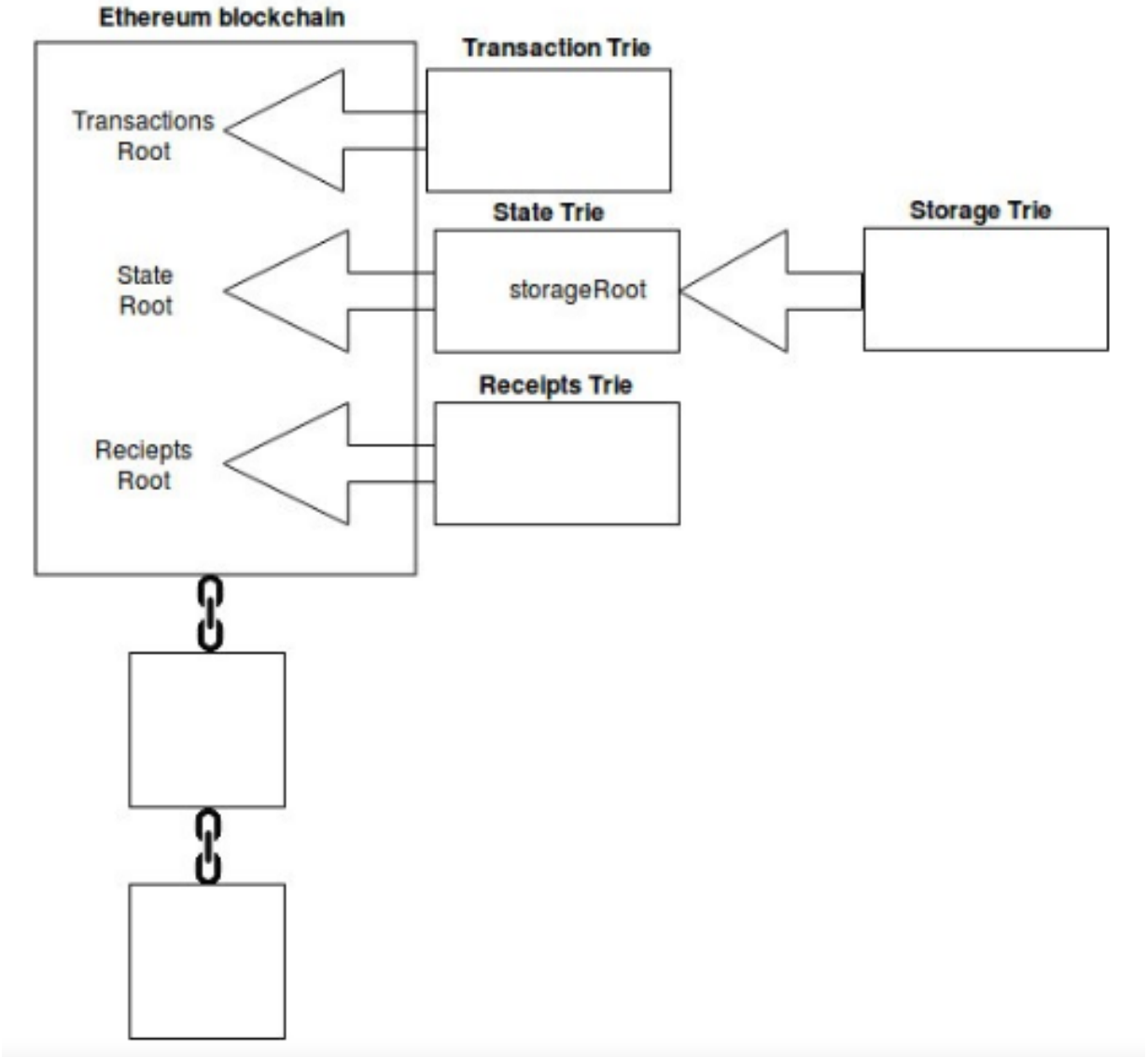


Figure 2.3: The four tries in an Ethereum block [T. McCallum, 2018]

- Transaction trie ensures the integrity of all the transactions in a block. Every block has its own transaction trie. It consists of the account nonce, gas price, gas limit, recipient, transfer value, transaction signature values, and account initialization [D. Brickwood, 2018].

- Receipts trie contains the information of each transaction separately and is a pool of four items: the cumulative gas used, the set of logs, the Bloom filter and the status code of the transaction.

- Storage trie is separate for each account and contains contract information.

Some of this data will be analyzed later.

## 2.8 Highlights of Ethereum roadmap

- January 2014: Ethereum project is introduced at the North American Bitcoin conference by Vitalik Buterin.
- July 2014: An Ethereum ICO was organized and attracted a lot of funding. A total of 60'102'216 ETH are sold for 31'591 BTC, which was worth $18'439'086 at that time [Easy Ethereum]. Out of these funds, Ethereum Switzerland GmbH was funded and developed the Ethereum project.
- May 2015: Ethereum launches Ethereum Olympic, a testing release where coins are not compatible with "real" ETH [A. Lewis, 2016].
- July 2015: Ethereum launches Ethereum frontier, a release broader to people who can mine ETH and build contracts.
- August 2015: Kraken becomes the first digital currency exchange platform to trade ether.
- March 2016: Homestead is the first "stable" Ethereum release and is classified as safe to participate.
- May/June 2016: DAO[6] raised $150 million using ICO. However, due to a gap in the smart contract coding that allowed the withdrawal of twice as much ether as invested, the Ethereum network implemented a hard fork. As a result, $50 million of Ether was stolen and the Ethereum blockchain split into Ethereum (ETH) and Ethereum Classic (ETC). This fork caused Ethereum to lose its immutability. After the fork, Ethereum attracted more users and it was mainly established as the common Ethereum [J. P. Buntinx, 2017].
- October 2016: Tangerine Whistle: another hard fork for Ethereum in the block 2'463'000 on $18^{th}$ of October 2016. This fork occurred in order to increase the gas prices and make them reflect the real computational complexity of some operations. As a result, any attack attempt is getting automatically more expensive and more difficult [Coinmama].
- February 2017: Enterprise Ethereum Alliance is established by a group of companies in order to make Ethereum suitable for big businesses. As a result, the concept of permissioned Ethereum is introduced.

[6]DAO (Decentralized Autonomous Organization) was the venture capital fund with no management structure that aimed to raise the money for Ethereum Dapps that were promising, by their own belief. The investments were distributed via DAO tokens [D. Vujicic and D. Jagodic and S. Randic, 2018].

- October 2017: Metropolis: Byzantium hard fork is the first half of Metropolis update. It occurred in the block 4'370'000 on October, 16-th 2017. It included changes to reduce the complexity of EVM and provide more flexibility to smart contract developers.
- February 2019: Metropolis: Constantinople hard fork is the second half of Metropolis update. It occurred in the block 7'280'000 on February, 28-th 2019.
- May 2019: Serenity is the last scheduled step of Ethereum and brings a big change in the consensus mechanism. Until then, the only acceptable consensus mechanism was proof of work. In this release, a hybrid mechanism (Casper) of proof of work and proof of stake will be in place. More details regarding this change can be found in the final chapter.

# Chapter 3

# Data Analysis Setup

## 3.1 Data acquisition

According to BitInfoCharts, the Ethereum database size is 212.17 GB. There is no central administrative system where the data can be downloaded from. One feature of Ethereum is that all data is public and available. The nodes, which share data among each other, store a copy of the data, while the network executes the Ethereum protocol, which defines the rules of interaction of nodes with each other and/or smart contracts over that network [A. Sokolowska, 2018].

A blockchain is an immutable, append-only distributed database [ETH events, 2018]. In order to analyze it and extract some useful information, it is important, not only to grasp the unique characteristics of this technology, such as decentralization, transparency and immutability, but also to understand what data is stored in the blockchain, how it looks like and how it is structured. Syncing an Ethereum node is a pain point for someone who wants to access Ethereum data. By downloading all the blocks, the whole chain is taken. However, as no transaction is executed, no account state is available and as a result no balances, nonces, smart contract code and other data are obtained. In order to acquire these, the state trie is needed, which requires a lot of time and processing memory. The schema of the Ethereum raw data is illustrated in fig. 3.1.

In August 2018, Google Big Query managed to download all the blocks' data in an SQL database, in real time. The first phase relies on Ethereum ETL - an open-source tool they developed in order to export the Ethereum blockchain into CSV or JSON files. It connects to an Ethereum node via its JSON RPC interface. The exported files are then moved to Google Cloud Storage, loaded into BigQuery, and finally verified. Then, the user can query the data in the BigQuery console or via API [E. Medvedev and A. Day, 2018]. Since all the data is available there and is updated every day, this source of data will be used as the main one for the analysis of the Ethereum data.

```
scala> ds.printSchema()
root
 |-- ethereumBlockHeader: struct (nullable = true)
 |     |-- parentHash: binary (nullable = true)
 |     |-- uncleHash: binary (nullable = true)
 |     |-- coinBase: binary (nullable = true)
 |     |-- stateRoot: binary (nullable = true)
 |     |-- txTrieRoot: binary (nullable = true)
 |     |-- receiptTrieRoot: binary (nullable = true)
 |     |-- logsBloom: binary (nullable = true)
 |     |-- difficulty: binary (nullable = true)
 |     |-- timestamp: long (nullable = false)
 |     |-- number: decimal(38,0) (nullable = true)
 |     |-- numberRaw: binary (nullable = true)
 |     |-- gasLimit: decimal(38,0) (nullable = true)
 |     |-- gasLimitRaw: binary (nullable = true)
 |     |-- gasUsed: decimal(38,0) (nullable = true)
 |     |-- gasUsedRaw: binary (nullable = true)
 |     |-- mixHash: binary (nullable = true)
 |     |-- extraData: binary (nullable = true)
 |     |-- nonce: binary (nullable = true)
 |-- ethereumTransactions: array (nullable = true)
 |     |-- element: struct (containsNull = true)
 |     |     |-- nonce: binary (nullable = true)
 |     |     |-- value: decimal(38,0) (nullable = true)
 |     |     |-- valueRaw: binary (nullable = true)
 |     |     |-- receiveAddress: binary (nullable = true)
 |     |     |-- gasPrice: decimal(38,0) (nullable = true)
 |     |     |-- gasPriceRaw: binary (nullable = true)
 |     |     |-- gasLimit: decimal(38,0) (nullable = true)
 |     |     |-- gasLimitRaw: binary (nullable = true)
 |     |     |-- data: binary (nullable = true)
 |     |     |-- sig_v: binary (nullable = true)
 |     |     |-- sig_r: binary (nullable = true)
 |     |     |-- sig_s: binary (nullable = true)
 |-- uncleHeaders: array (nullable = true)
 |     |-- element: struct (containsNull = true)
 |     |     |-- parentHash: binary (nullable = true)
 |     |     |-- uncleHash: binary (nullable = true)
 |     |     |-- coinBase: binary (nullable = true)
 |     |     |-- stateRoot: binary (nullable = true)
 |     |     |-- txTrieRoot: binary (nullable = true)
 |     |     |-- receiptTrieRoot: binary (nullable = true)
 |     |     |-- logsBloom: binary (nullable = true)
 |     |     |-- difficulty: binary (nullable = true)
 |     |     |-- timestamp: long (nullable = false)
 |     |     |-- number: decimal(38,0) (nullable = true)
 |     |     |-- numberRaw: binary (nullable = true)
 |     |     |-- gasLimit: decimal(38,0) (nullable = true)
 |     |     |-- gasLimitRaw: binary (nullable = true)
 |     |     |-- gasUsed: decimal(38,0) (nullable = true)
 |     |     |-- gasUsedRaw: binary (nullable = true)
 |     |     |-- mixHash: binary (nullable = true)
 |     |     |-- extraData: binary (nullable = true)
 |     |     |-- nonce: binary (nullable = true)
```

Figure 3.1: Ethereum raw data

## 3.2 Data structuring in Google Bigquery Ethereum

Google Bigquery separates the Ethereum data in 7 tables: blocks, contracts, logs, token_transfers, tokens, traces, and transactions. In tables 3.1 to 3.7, all the available data fields, together with their data types are presented. There are some data field overlaps in the tables, such as the block number and the block timestamp. The tables are structured this way in order to be easy for analysis.

Table 3.1: Token transfers table

| Fields | Data type | Mandatory | Description |
|---|---|---|---|
| token_address | string | + | ERC20 token address |
| from_address | string | | Address of the sender |
| to_address | string | | Address of the receiver |
| value | string | | Amount of tokens transferred (ERC20) / id of the token transferred (ERC721) |
| transaction_hash | string | + | Transaction hash |
| log_index | integer | + | Log index in the transaction receipt |
| block_timestamp | timestamp | + | Timestamp of the block where this transfer was in |
| block_number | integer | + | Block number where this transfer was in |
| block_hash | string | + | Hash of the block where this transfer was in |

Although the tables consists of all these data fields, only some of those will be used for the main analysis. Here is a broader explanation of the data that will be mostly analyzed in chapter 4:

**to_address:** The recipient address of an Ethereum transaction. This could be either an external owned account or a contract account. It is represented with a 160-bit address, unless it is null, which would mean that the transaction represents a contract creation. It is the result of a hash function.
**from_address:** The initiator address of an Ethereum transaction. It can only be an exter-

Table 3.2: Tokens table

| Fields | Data type | Mandatory | Description |
|---|---|---|---|
| address | string | + | The address of the ERC20 token |
| symbol | string | | The symbol of the ERC20 token |
| name | string | | The name of the ERC20 token |
| decimals | string | | The number of decimals the token uses |
| total_supply | string | | The total token supply |

Table 3.3: Contracts table

| Fields | Data type | Mandatory | Description |
|---|---|---|---|
| address | string | + | Address of the contract |
| bytecode | string | | Bytecode of the contract |
| function_signhashes | array of string | | 4-byte function signature hashes |
| is_erc20 | boolean | | True when the contract is an ERC20 contract |
| is_erc721 | boolean | | True when the contract is an ERC721 contract |
| block_timestamp | timestamp | + | Timestamp of the block where this contract was created |
| block_number | integer | + | Block number where this contract was created |
| block_hash | string | + | Hash of the block where this contract was created |

nal owned account, as EOA are the only ones who can initiate transactions.
**miner:** The address of the beneficiary of the reward for any block validation. These are miner addresses and will help us identify the activity of miners and separate them from the other external addresses.
**value:** A scalar value equal to the amount of Wei that the sender transfers to the recipient.
**input:** The data sent during a transaction, which is also called "message". When an ex-

ternal account executes a transaction to another external account, the input could be a message which the sender wants to send to the repicient. Otherwise, the input data is usually empty. Another type of message is the one that is sent to a contact. The input there is the code that should be executed to the called contract.
**gas_limit:** The maximum number of computational steps or the maximum amount of gas the transaction can spend. If a transaction needs more gas in order to be executed, it will fail with an "out of Gas" status. This is an important element for a transaction to be executed properly. The gas limit protects smart contracts from infinite loops. Lowering the gas limit does not mean that less gas will be spent.
**gas_used:** Despite the gas limit, a transaction will use a specific amount of gas to be executed. This is the actual amount of Gas spent. For example, a simple Ether transfer requires a minimum of 21,000 gas units. Any other operation costs more than 21,000 gas units [H. Kenneth, 2018]. One can find relative costs of abstract operations in the Ethereum yellow paper [G. Wood]. In case the gas used is less than the gas limit, the remaining gas will not be deducted from the account, but will be returned to the owner.
**gas_price:** The Gas price, in ETH, that the sender defines at transaction creation and is willing to pay for each computational step [Ł. Żuchowski, 2017]. The higher it is, the bigger the incentive for miners to prioritize the transaction. However, some recommended gas prices can be found in the ETH gas station [ETH Gas Station]. For example, a normal price is 4.8 Gwei for a fast execution of less than two minutes. A safe slow-time execution gas price is 1 Gwei.
**output:** The final bytecode of the contracts created. It constitutes the rules of how a specific contract works. It will be used to identify the identical contracts.
**call_type:** The indicator of what type of call one address does to another. It will be used to identify the nature of the different transactions.
**trace_address:** This is null for top level traces, i.e. those corresponding to transactions. Thus, they are created by external accounts. On the other side, it is not null when it is a message call between two contracts. It will be used in the categorization of contracts into two categories: the user-created ones and the contract-created ones.
**status:** This defines if the transaction was successful (status 1) or failed (status 0).

Table 3.4: Blocks table

| Fields | Data type | Mandatory | Description |
|---|---|---|---|
| timestamp | timestamp | + | The timestamp for when the block was collated |
| number | integer | + | The block number |
| hash | string | + | Hash of the block |
| parent_hash | string | | Hash of the parent block |
| nonce | string | + | Hash of the generated proof-of-work |
| sha3_uncles | string | | SHA3 of the uncles data in the block |
| logs_bloom | string | | The bloom filter for the logs of the block |
| transactions_root | string | | The root of the transaction trie of the block |
| state_root | string | | The root of the final state trie of the block |
| receipts_root | string | | The root of the receipts trie of the block |
| miner | string | | The address of the beneficiary to whom the mining rewards were given |
| difficulty | numeric | | Integer of the difficulty for this block |
| total_difficullty | numeric | | Integer of the total difficulty of the chain until this block |
| size | integer | | The size of this block in bytes |
| extra_data | string | | The extra data field of this block |
| gas_limit | integer | | The maximum gas allowed in this block |
| gas_used | integer | | The total used gas by all transactions in this block |
| transaction_count | integer | | The number of transactions in the block |

Table 3.5: Transactions table

| Fields | Data type | Mandatory | Description |
|---|---|---|---|
| hash | string | + | Hash of the transaction |
| nonce | integer | + | The number of transactions made by the sender prior to this one |
| transaction_index | integer | + | Integer of the transactions index position in the block |
| from_address | string | + | Address of the sender |
| to_address | string | | Address of the receiver. Null when it's a contract creation transaction |
| value | numeric | | Value transferred in Wei |
| gas | integer | | Gas provided by the sender |
| gas_price | integer | | Gas price provided by the sender in Wei |
| input | string | | The data sent along with the transaction |
| receipt_cumulative_gas_used | integer | | The total amount of gas used when this transaction was executed in the block |
| receipt_gas_used | integer | | The amount of gas used by this specific transaction alone |
| receipt_contract_address | string | | The contract address created, if the transaction was a contract creation, otherwise null |
| receipt_root | string | | 32 bytes of post-transaction stateroot (pre Byzantium) |
| receipt_status | integer | | Either 1 (success) or 0 (failure) (post Byzantium) |
| block_timestamp | timestamp | + | Timestamp of the block where this transaction was in |
| block_number | integer | + | Block number where this transaction was in |
| block_hash | string | + | Hash of the block where this transaction was in |

Table 3.6: Traces table

| Fields | Data type | Mandatory | Description |
|---|---|---|---|
| transaction_hash | string | | Transaction hash where this trace was in |
| transaction_index | integer | | Integer of the transactions index position in the block |
| from_address | string | | Sender's address, null when trace_type is genesis or reward |
| to_address | string | | *See footnote*[1] |
| value | numeric | | Value transferred in Wei |
| input | string | | The data sent along with the message call |
| output | string | | The output of the message call, bytecode of contract when trace_type is create |
| trace_type | string | + | One of call, create, suicide, reward, genesis, daofork |
| call_type | string | | One of call, callcode, delegatecall, staticcall |
| reward_type | string | | One of block, uncle |
| gas | integer | | Gas provided with the message call |
| gas_used | integer | | Gas used by the message call |
| subtraces | integer | | Number of subtraces |
| trace_address | string | | Comma separated list of trace address in call tree |
| error | string | | Error if message call failed (no top-level trace errors) |
| status | integer | | Either 1 (success) or 0 (failure) |
| block_timestamp | timestamp | + | Timestamp of the block where this trace was in |
| block_number | integer | + | Block number where this trace was in |
| block_hash | string | + | Hash of the block where this trace was in |

[1] Address of the receiver if trace_type is call, address of new contract or null if trace_type is create, beneficiary address if trace_type is suicide, miner address if trace_type is reward, shareholder address if trace_type is genesis, WithdrawDAO address if trace_type is daofork

Table 3.7: Logs table

| Fields | Data type | Mandatory | Description |
|---|---|---|---|
| log_index | integer | + | Integer of the log index position in the block |
| transaction_hash | string | + | Hash of the transactions this log was created from |
| transaction_index | integer | + | Integer of the transactions index position log was created from |
| address | string | | Address from which this log originated |
| data | string | | Contains one or more 32 Bytes non-indexed arguments of the log |
| topics | array of string | | Indexed log arguments (0 to 4 32-byte hex strings)[1] |
| block_timestamp | timestamp | + | Timestamp of the block where this log was in |
| block_number | integer | + | The block number where this log was in |
| block_hash | string | + | Hash of the block where this log was in |

# Chapter 4

# Data Analytics

## 4.1 Methodology

In chapters 1 to 3, the distributed ledger technology and the blockchain technology were analyzed as concepts and the focus was put mostly on the Ethereum structure. The Ethereum blockchain contains a lot of data, not only for the pure currency transactions, but also for the smart contracts that operate in the network and change the state of Ethereum.

In the main analysis, the focus will be initially given to a comparison of how the Ethereum is being used in general over time and to its "demographics" as of today. Then, in the second phase, the number of transactions and the total value of Ether will be used as common denominator in order to compare the different external accounts, the different contracts, and the different tokens. Moreover, the top external accounts, contracts and tokens will be identified and compared with each other, in an attempt to define the reason why these accounts present a large number of transactions or value of Ether. Finally, a closer look will be taken at the contracts analysis, and more particularly, their lifecycle and how they are used. Depending on these comparison results, clusters may be built.

The queries on the SQL Google Bigquery Ethereum database that are used for this analysis are provided in part II. Moreover, [Etherscan.io], a Block Explorer, Search, API and Analytics Platform for Ethereum, will be used for identifying specific details for different addresses. This platform provides some analytics as well, concentrating on the "demographics" data and not on the analysis of the usage and comparison of the different addresses.

## 4.2 Generic Analytics on Ethereum

Having described in detail how the Ethereum blockchain works and what types of data are available for analysis, we will first try to analyze the "demographics" of Ethereum and provide a snapshot of its blockchain as of $20^{th}$ of May, 2019.


Figure 4.1: Daily number of transactions in the Ethereum network. The peak was 1'349'890 transactions on the $4^{th}$ of January 2018.


Figure 4.2: Daily number of transactions in the Ethereum network (logarithmic scale)

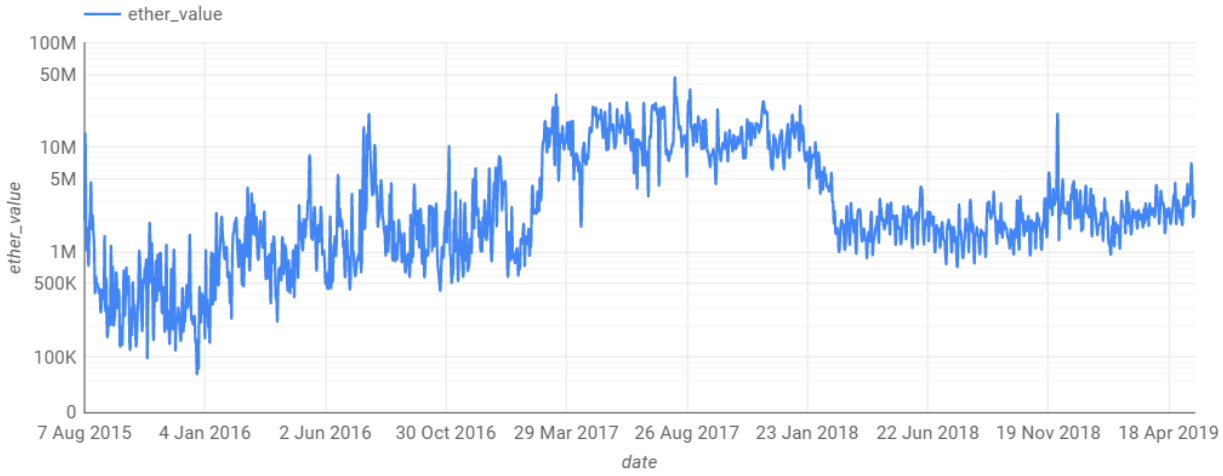


Figure 4.3: Daily transferred ether value (value measured in ether, the respective currency of the Ethereum network

### 4.2.1 Number of transactions (traffic)

The Ethereum network has processed in total more than 450'528'110 transactions as of $20^{th}$ of May. In fig. 4.1, the time series of the number of transactions per day are illustrated. The daily record for the maximum number of transactions on a single day can be found on $4^{th}$ of January 2018, which is 1'349'890 transactions. The longest continuous increase of transactions 195%, found between November 2017 and January 2018. However, there is a sharp decrease after January 2018 (134% decrease) and a final stabilization at around 750'000 daily transactions, on average. This is still more than the total average number of daily transaction since the genesis of the blockchain, which is roughly 326'383, because of the long tail of small number of transactions for the first 2 years of Ethereum. Although the future of Ethereum has been doubted during the last months, it seems that it attracts the interest of users since February 2019, while the number of daily transactions has increased to around 900'000 in May 2019. This number has not been reached since May 2018.

### 4.2.2 Volume of transactions

Apart from the number of transactions, it is very interesting to see what the total value of ether is in these transactions. There may be a lot of transactions, but of small ether value, or the other way around, meaning only a few transactions where a large amount of ether is transferred. The total value of transactions is 6'628'592'258 ETH.

The daily total value of Ether can be found in fig. 4.3. This is the recorded ether value in all the transactions. Between March 2017 and January 2018, there is an average of 15'000'000 ETH, transferred per day. There is a peak of approximately 45'000'000 ETH in $11^{th}$ of August 2017. Before March 2017 and after January 2018, the daily total value of

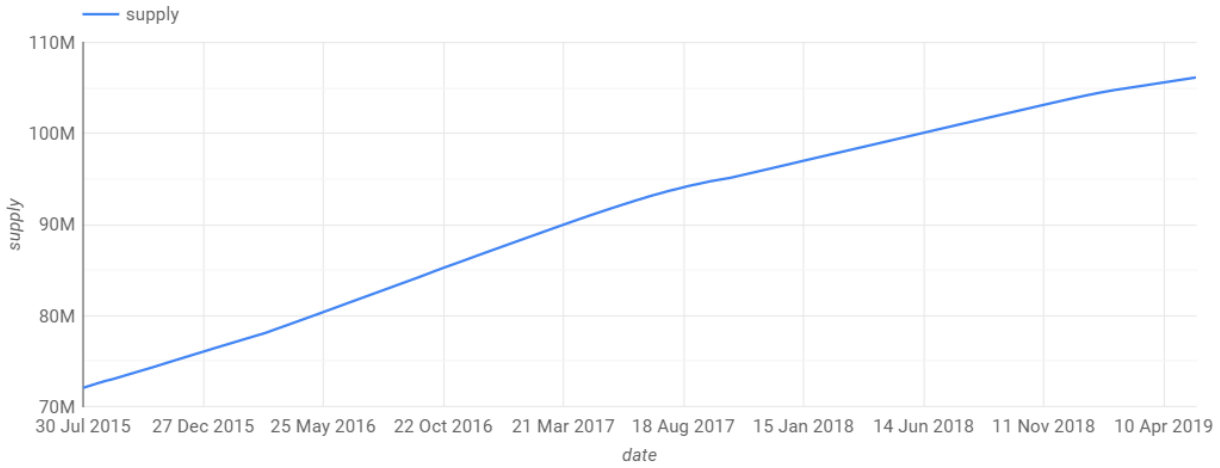


Figure 4.4: Ether supply growth. The supply consists of the initial ether supply in the system when it was created and the ether generated as a reward for any mining activity.

the transactions is moving around 2'500'000 ETH, with only a few exceptions.

Figure 4.4 shows the ether supply and its growth rate. The supply consists of the initial ether supply in the system when it was created and the ether generated as reward for mining activity. When Ethereum first launched, it "pre mined" 72'000'000 ether. Since then, 34'000'000 more ether have been generated, as rewards for main block and uncle block miners. As a result, the ether supply increases around 7.3% per year. On $16^{th}$ of October 2017, the ether supply growth rate started decreasing. This is associated with the block reward reduction from 5 ether to 3 ether when the Byzantium hard fork was applied. A second reduction in the mining rewards, from 3 ether to 2 ether, occurred on $16^{th}$ of January 2019 during the Ethereum hard fork, called Constantinople, at block 7'080'000 [H. Kenneth, 2018]. This reduction delivers the purpose of reducing the inflation.

### 4.2.3 Gas used

Another important characteristic is the gas used in these transactions. As mentioned earlier, what distinguishes Ethereum from other cryptocurrencies is the ability to create DAPPs and smart contracts. The more complex a transaction is, e.g. the ones that address to smart contracts and DAPPs, the more gas is consumed. As seen earlier, the minimum amount of gas used is 21'000 gas for a simple ether transfer. However, the average gas used is 3'217'249 gas. Figure 4.5 shows how the daily gas used in transactions is continuously increasing. Taking into consideration the reduction in the number of transactions that occurred after January 2018, one can conclude that the Ethereum network started operating, not only as a simple ether transfer network, but also as a contract execution network, since the latter type of transactions costs more in terms of gas. Later, using more data from the network, the

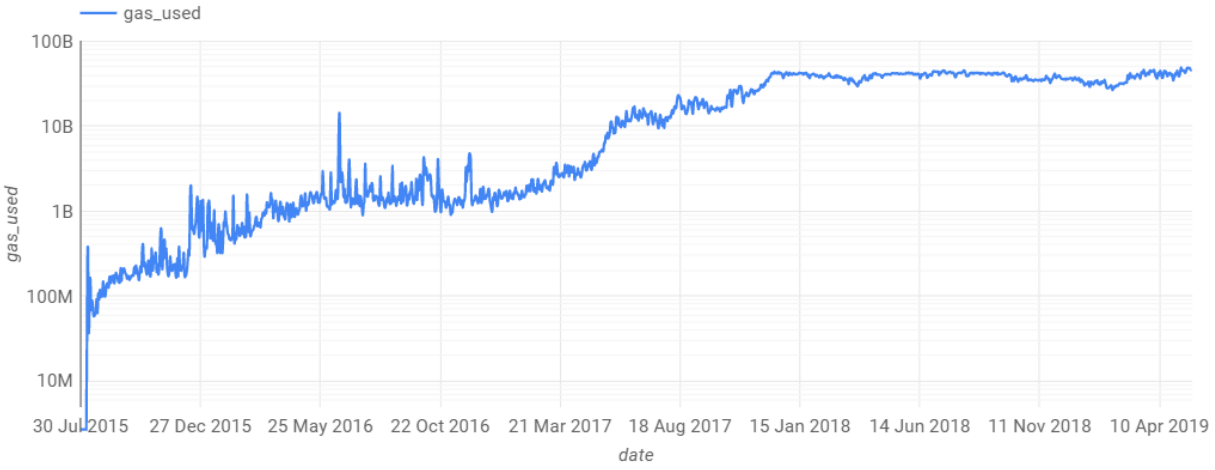


Figure 4.5: Daily gas consumption in the Ethereum network. It represents the sum of ether spent each day. The y axis is in logarithmic scale.

different contracts that contribute also most to this gas increase will be further analyzed.

### 4.2.4 Gas price

As explained earlier, the gas price is set by the initiator of a transaction. The higher the price is, the more probable for the transaction to be prioritized higher against others. In general, fig. 4.6 shows an average gas price of around 20B Wei[1]. This average takes into consideration only the different gas prices of the successful transactions. There is no reason to include gas prices of transactions that were not executed, since this would violate the real level of gas evaluation.

In February 2019, there is a peak in the daily average gas price, close to 400B wei . Looking further into the data, there was a transaction with 100'000'000B wei gas price on that day. This is the highest gas price ever recorded in the Ethereum network. A deeper check would let us find out that the value of the transactions and the gas price are identical; that is 100'000'000B wei. The most possible explanation is that the developer confused the two data fields. As table 4.1 shows, the same address executed more transactions like this where the gas price and the value of the transaction were identical. Unfortunately, an error like this is irreversible in the current Ethereum infrastructure.

### 4.2.5 Address growth

For any network, the users constitute a dominant part of its success. Users in the Ethereum network are not only individuals, but also contracts. In fig. 4.7, the cumulative address

[1]Only gas price data after Byzantium is considered (see section 2.8)

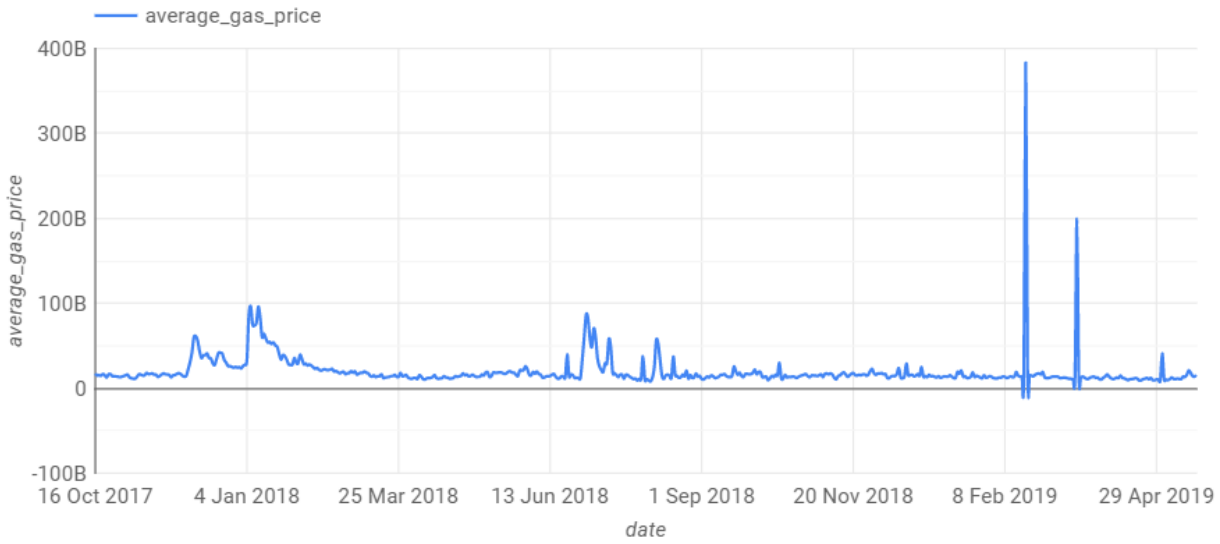


Figure 4.6: Daily average price of gas, measured in wei. The ratio of eei- ether is 1 ether per 1'018 wei. The representation in ether would be a line close to zero. Therefore the fluctuations are better visualized in the wei scale.

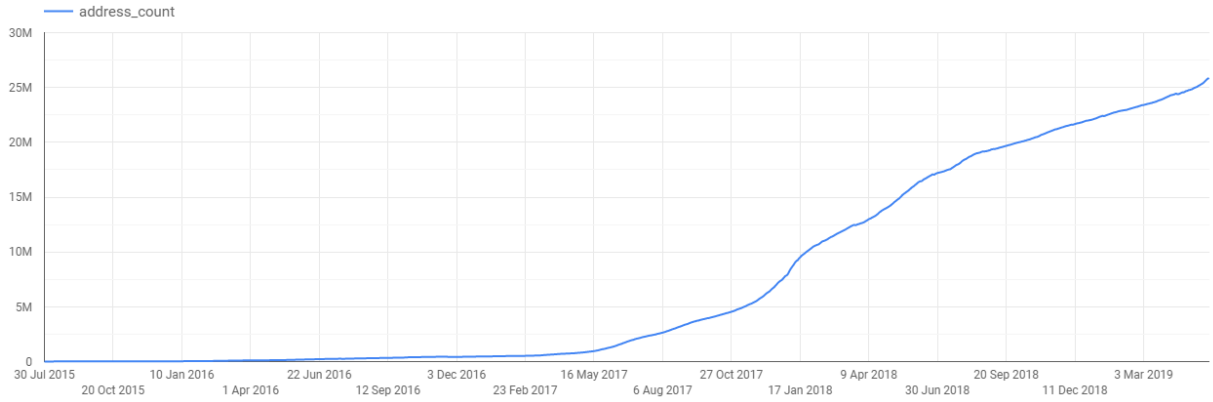


Figure 4.7: Cumulative number of unique addresses in the network with a balance greater than zero. As the balance of the addresses is taken into consideration, some contracts are not calculated here.

Table 4.1: All the high in gas price transactions of 19.02.2019

| Row | from_address | Value (wei) | gas_price (wei) |
|---|---|---|---|
| 1 | 0x587ecf600d304f831201c30ea0845118dd57516e | $2x10^{16}$ | $2x10^{16}$ |
| 2 | 0x587ecf600d304f831201c30ea0845118dd57516e | $2x10^{16}$ | $4x10^{16}$ |
| 3 | 0x587ecf600d304f831201c30ea0845118dd57516e | $1x10^{16}$ | $1x10^{16}$ |
| 4 | 0x587ecf600d304f831201c30ea0845118dd57516e | $2x10^{16}$ | $2x10^{16}$ |
| 5 | 0x587ecf600d304f831201c30ea0845118dd57516e | $1x10^{17}$ | $1x10^{17}$ |

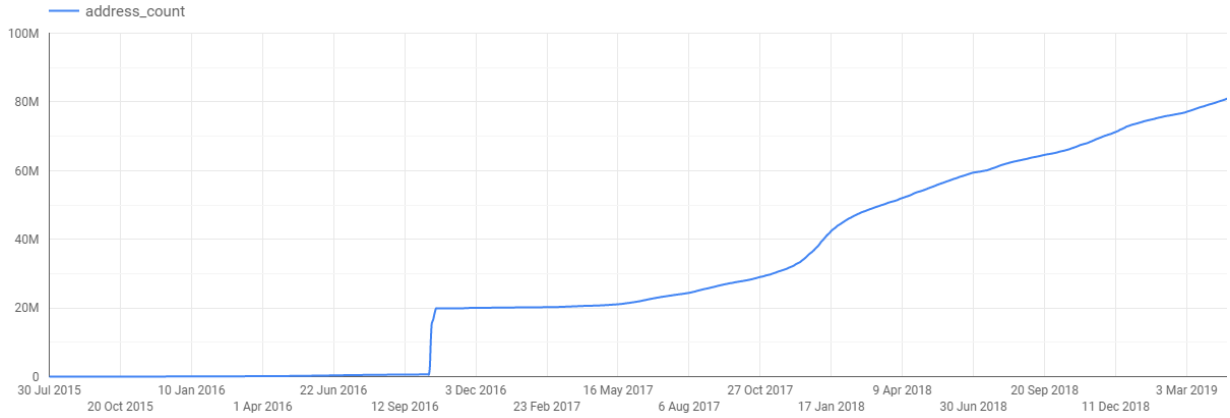


Figure 4.8: Cumulative number of unique addresses in the network. The increase of the number of addresses is obvious, as well as the strange behaviour.

growth can be found. Since the balance of the addresses is taken into consideration, some contracts are not considered here. As of $20^{th}$ of May 2019, there are 25 million addresses in the Ethereum network with a balance larger than zero. In almost a year, from 2018, the number of distinct addresses has been doubled.

However, the same graph, considering also addresses with a zero balance, can be a different story. In fig. 4.8, a sharp increase of the number of addresses by 20 million is observed. Indeed, on $22^{nd}$ of September 2016, almost 20'000'000 addresses were suddenly created. That was the result of a DDos attack. A hacker took advantage of an underpriced upcode and created around 20'000'000 useless Ethereum accounts. This led to the creation of millions of transaction traces, making data analysis more difficult. At that specific time, the network was slowing down, making it more time consuming for miners and nodes to process blocks [Day and Medvedev, 2018]. Thereafter, the Ethereum network underwent a hard fork at block number 2'463'000. Finally, it can be stated that the total number of addresses is approximately 60M, given the 80M transactions minus the 20M useless addresses.

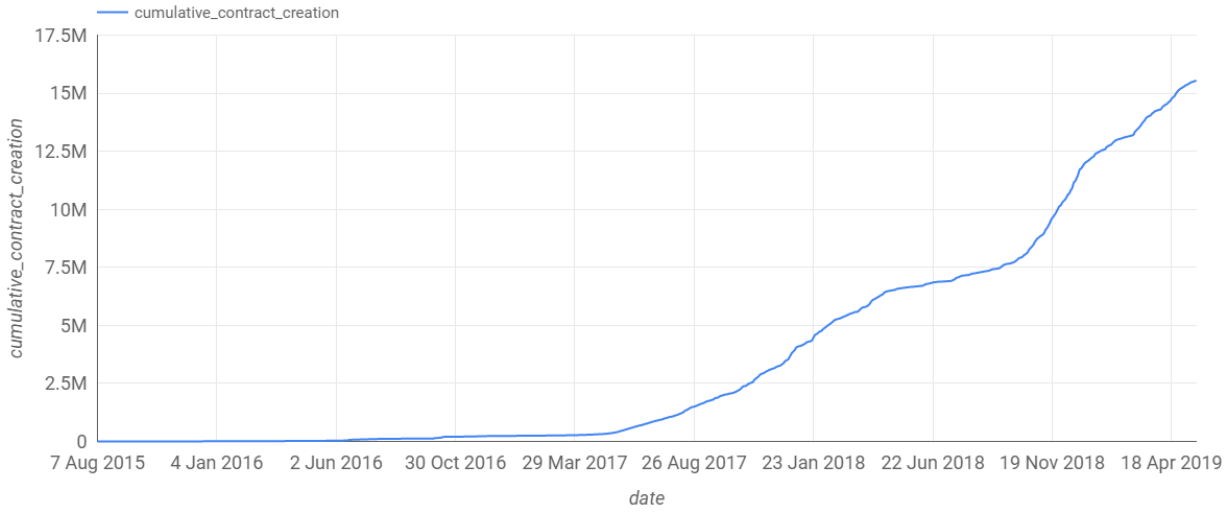


Figure 4.9: Cumulative creation of smart contracts

The different types of addresses are displayed in table 4.2. The total number of addresses is calculated for addresses that have a balance, strictly greater than zero, but also for the whole universe of addresses. The approximate calculation, based on figs. 4.7 to 4.8, can also be verified in table 4.2. Summing up the total number of contracts and senders, where balance is greater or equal to zero, gives a total of 60M addresses. The number of miners is relatively very small to be taken into consideration in the approximate calculation.

Table 4.2: Number of addresses, per type of address

| **Addresses** | **Where balance > 0** | **Where balance >= 0** |
|---|---|---|
| **All** | 25'869'427 | 83'930'036 |
| **Contracts** | 497'999 | 15'571'524 |
| **Miners** | 3'114 | 4'770 |
| **Senders** | 22'448'669 | 46'239'732 |

### 4.2.6 Contracts

In total, there are 15'571'524 addresses that represent contracts. The growth of such addresses is displayed in fig. 4.9. This data is derived from the Google Big Query related table where all the contract details are stored. Later, the contracts that are directly created from external users will be separated from the ones that are created indirectly from external users, i.e. from a contract generating another contract. In April 2016, the first boost of

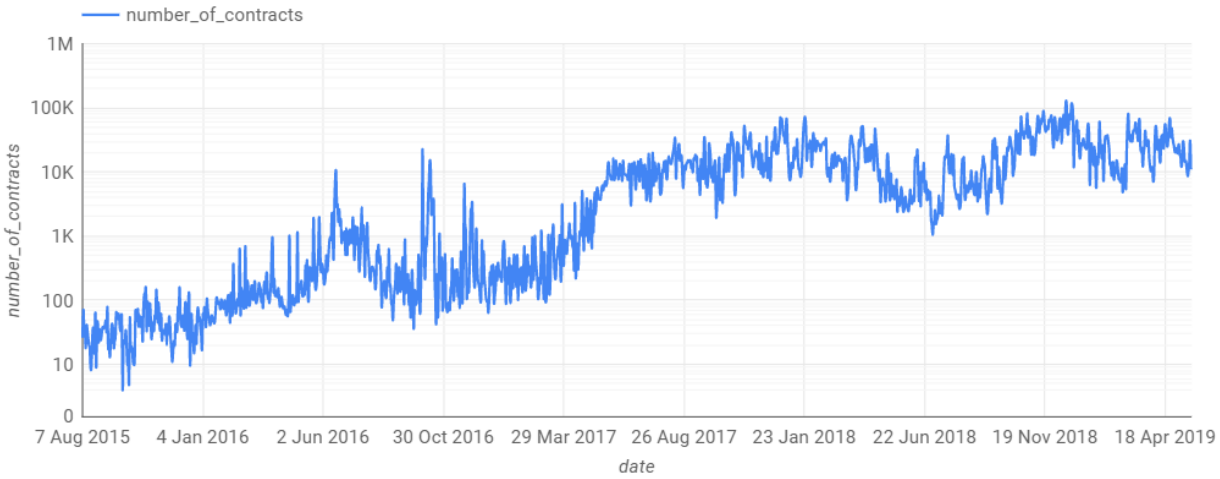


Figure 4.10: Daily creation of contract addresses (logarithmic scale)

smart contracts can be found. After January 2018, there are some picks and after May 2018, there is a small decrease.

However, the contact creation does not follow the pattern of transactions of high fluctuations. In the logarithmic scale, which can be found in fig. 4.10, the contract creation seems to be more constant and it is increasing over time. This means that users have understood over time the main functionality of Ethereum and its utility.

### 4.2.7 Allocation of addresses

In figs. 4.7 to 4.8, the Ethereum addresses growth for external users' addresses is displayed as of $20^{th}$ of May 2019. In fig. 4.11, the daily allocation of these addresses among the different types of addresses is displayed. What is visible is that the allocation of the total addresses has changed since the very beginning of Ethereum. Initially most of the addresses were mainly external users and very few contracts; through time, contracts gradually gained more ground. Miner addresses were always representing a small subset of the total addresses, which is also getting smaller, as more addresses of the other two categories are created.

### 4.2.8 ETH price and market cap

The fluctuation of the ETH price in USD, as reported in [CoinMarketCap], is shown in fig. 4.12. The ETH price follows the same pattern with the number of transactions. After January 2018, which is when the highest price is recorded, Ethereum lost 93% of its value, with \$84/ETH to be its lowest price. Since February 2019, Ethereum has made a remarkable return, stabilizing at around \$250/ETH, on average. The total market cap of Ethereum as

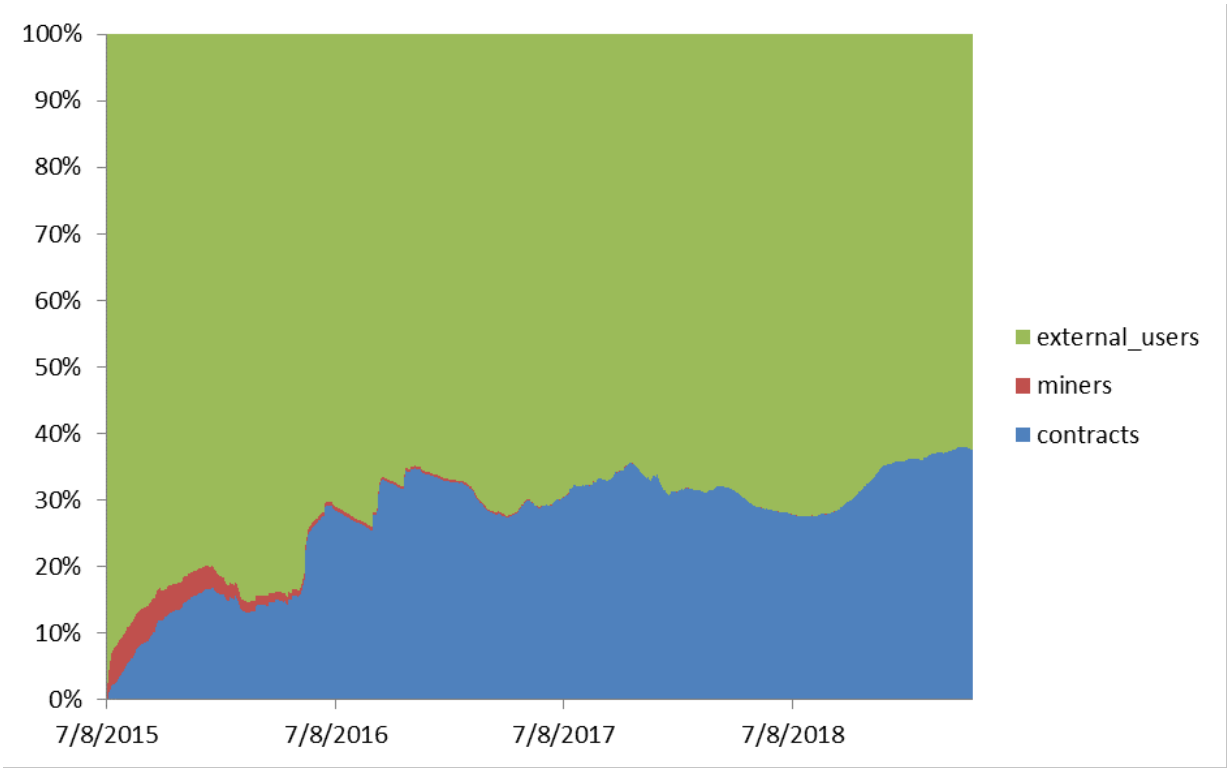


Figure 4.11: Daily allocation of total number of addresses

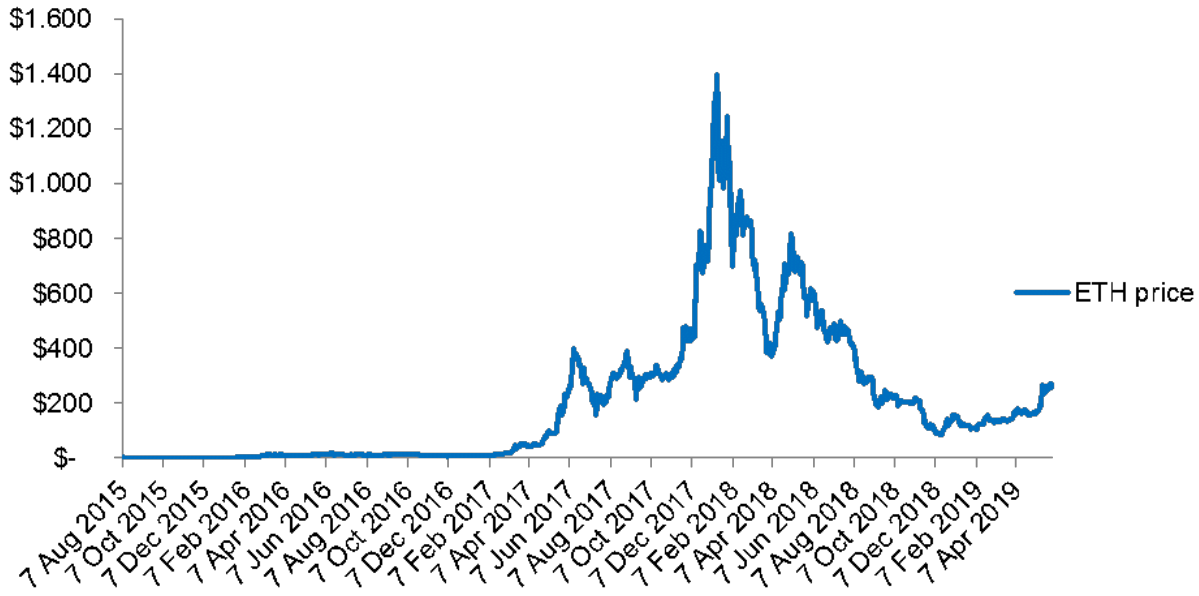


Figure 4.12: Daily price of ether in USD. The higher price was approximately at $1'400/ETH when the whole cryptocurrency market was evaluated relatively high in January 2018.

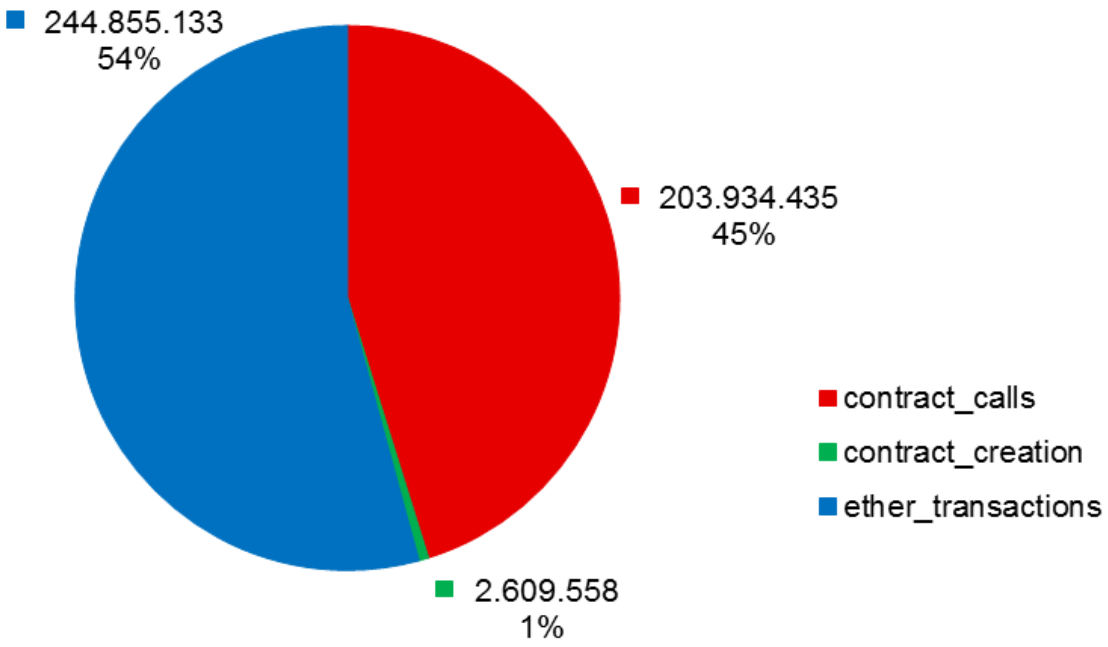


Figure 4.13: Allocation of the three different types of transactions (contract calls, ether transfer, and contract creation)

of $20^{th}$ of May 2019 is $26'736'046'292.

## 4.3 Analytics focusing on traffic (number of transactions)

The transactions are one of the main sources of answering many questions about the Ethereum activity. They display all the activity that is happening from external accounts to other external accounts or contracts. A transaction between two external accounts can be only transfer of value. On the other side, a transaction from an external account to a contract account activates the function of the contract performing different actions such as creating new contracts, writing to internal storage, transfering tokens, performing some actions, etc. [H. Kenneth, 2018].

From all the transactions in the network, we can have 3 types of transactions:

- **Ether transfer:** An external account can transfer Ether to another account or a contract. This kind of transaction is very similar to a Bitcoin transfer. The metadata of this transaction are the sender's address, the recipient's address, the timestamp and the value of the Ether sent.

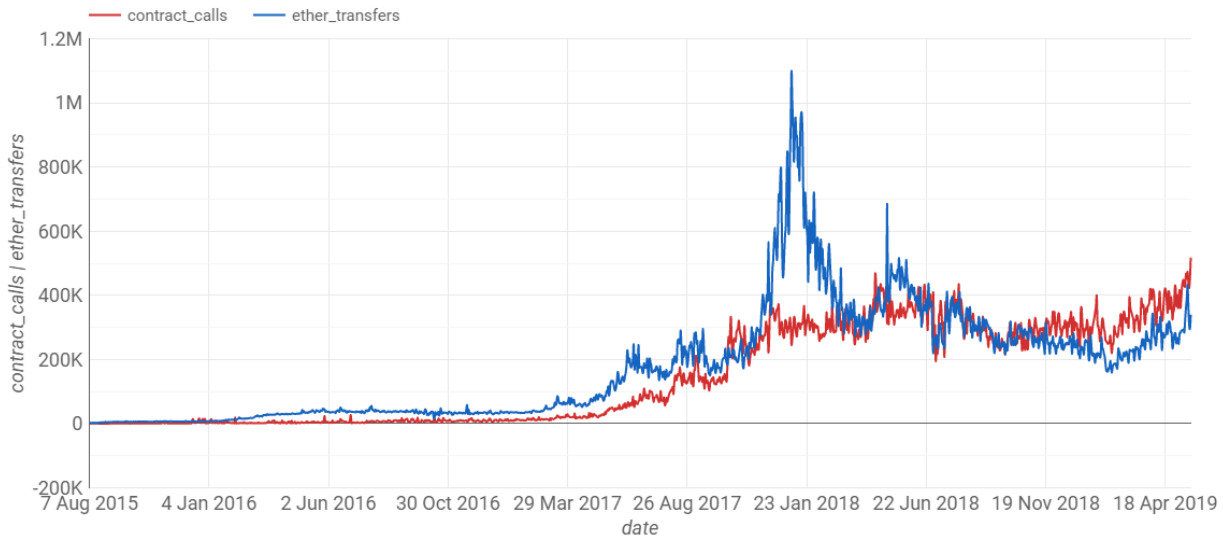


Figure 4.14: Comparison of the daily number of ether transfer transactions and contract calls

- **Contract creation:** An external account can create a contract by transferring ether to a zero recipient's account. Then, a new contract is created. The metadata of this transaction are the sender's address, the timestamp and the input.
- **Contract call:** An external account can call a contract when the account intends to execute one or more functions of a contract. The input data contains all the instructions related the execution of the contract. The metadata of this transaction are the sender's address, the recipient's address, the timestamp and the input.

Figure 4.13 shows that 54% of the transactions are Ether transfers. However, almost 45% of the transactions are calling a contract in order to execute a function of the contract. This indicates that a significant fraction of the activity in Ethereum uses the smart contract features and that it is not used only for currency transfers. Regarding the contract creation calls, it can be observed that only 1% of the transactions are of this type. At this point, it is important to mention that this number only refers to contracts created by external users, as transactions can be executed only from external users, and not from contracts. Later, the allocation of the different sources of contracts creation will be analyzed deeper.

During the last 4 years of Ethereum network usage, the allocation has changed a lot. In fig. 4.14, the daily number of transactions that are simple ether transactions is compared to the one of transactions that are contract calls. From the initialization of the Ethereum network until December 2018, there is a gradual increase for both types of transactions with a ratio of 2:1 in favour of Ether transactions. The reason is that Ethereum launched after the successful story of Bitcoin and the generic evolution of digital currencies. Users

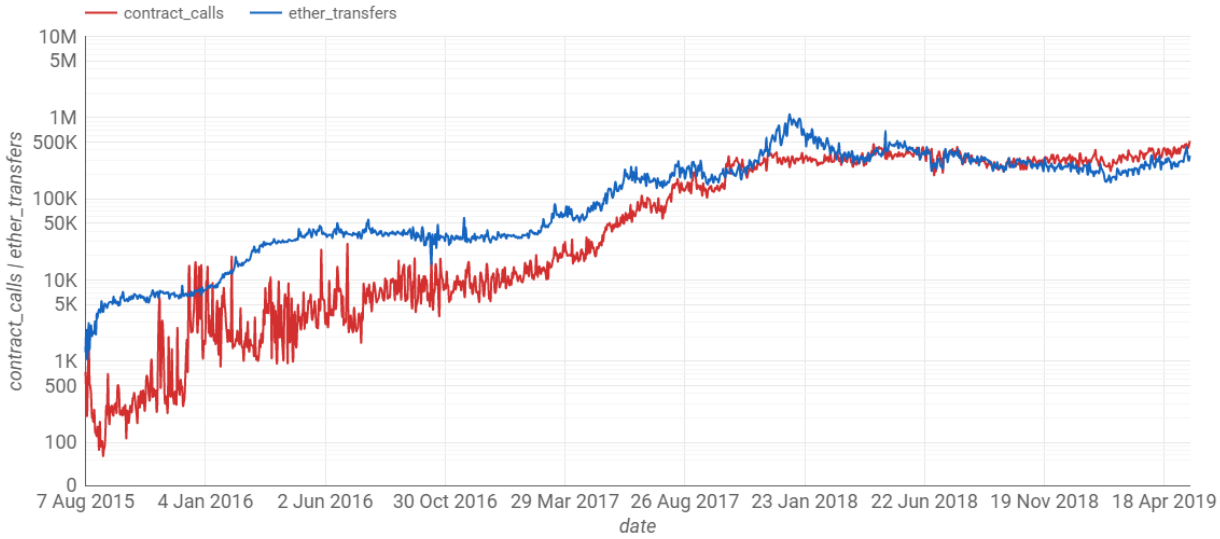


Figure 4.15: Comparison of the daily number of ether transfer transactions and contract calls (logarithmic scale). A dramatic increase in contract calls can easily be observed.

were considering Ethereum another cryptocurrency for peer to peer transactions or a speculative investment opportunity.

After December 2018, although contract call transactions continue to increase gradually, there is a sharp increase of Ether transactions that constituted 78% of total transactions in January 2018. This was the time when all the cryptocurrencies were thriving and were used as investment opportunities. However, the mid-2018 has seen the number of Ether transactions decreasing dramatically and reaching the number of contract calls. This was the time when Ethereum lost 45% of its value in only two weeks. In the same period, the number of contract calls remained the same with some small fluctuations. After November 2018, most of the transactions are contract calls, which overpassed the number of simple ether transactions.

## 4.4 Analytics focusing on volume (value of Ether)

Section 4.2 shows that there is a total of 106M ether supplied in the network, while all the Ethereum transactions have circulated more than 6.6B ether, as of $20^{th}$ of May 2019. The daily aggregated ether value of these transactions, both for simple ether transfers, as well as for contract calls, can be found in fig. 4.16.

Most of the ether value is circulated in simple ether transfers, from account to account. For almost a year, from 2017 to 2018, the average volume of ether transfers is around 10M

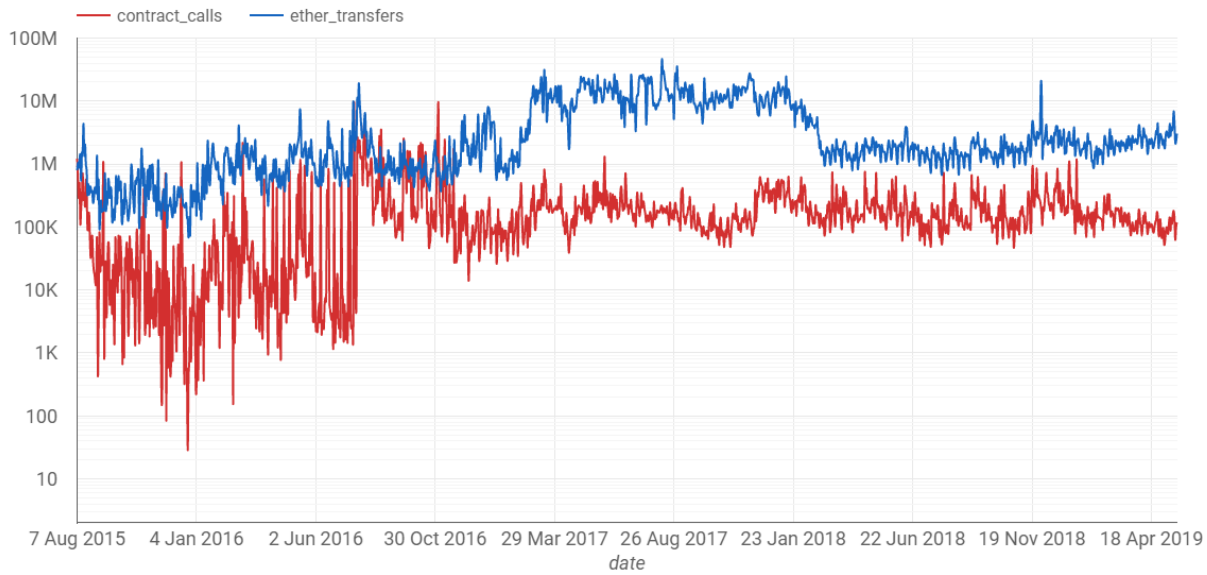


Figure 4.16: Daily volume of ether transferred in the ether transfer transactions and in the contract calls. In both of them, the value of each type of transaction is measured (logarithmic scale)

ether per day. This has been stabilized to around 5M as of $20^{th}$ of May 2019. Respectively, high fluctuations are found in the daily ether value in the contract calls during the first period. A possible reason is that initially contract creation process was in an experimental level where developers were trying to understand what type of contracts makes sense to have and there was a ether transfer in both simple ether transfers and contract calls. Later, this number is stabilized to approximately 100-300K and reflects the purchase of tokens in contracts.

## 4.5 Analytics focusing on the accounts

Furthermore, some observations are made regarding the activity of the separate accounts, as measured by the ether volume and the number of transactions in the Ethereum network as of the $20^{th}$ of May 2019.

In fig. 4.17, the ether balance distribution among the different addresses in Ethereum is displayed. It is logarithmically scaled in both axes, as ether balances cannot be adequately illustrated in a linear approach. There are addresses that own just some 0.001 ether and others that own more than 2M ether. Taking into consideration a total of around 25M addresses (external accounts and contract accounts with a balance greater than zero), it is concluded that the top 10 accounts hold 12% of the total ether and 80% of the total ether is held by the top 6k addresses. This is a strong representation of the pareto general rule of

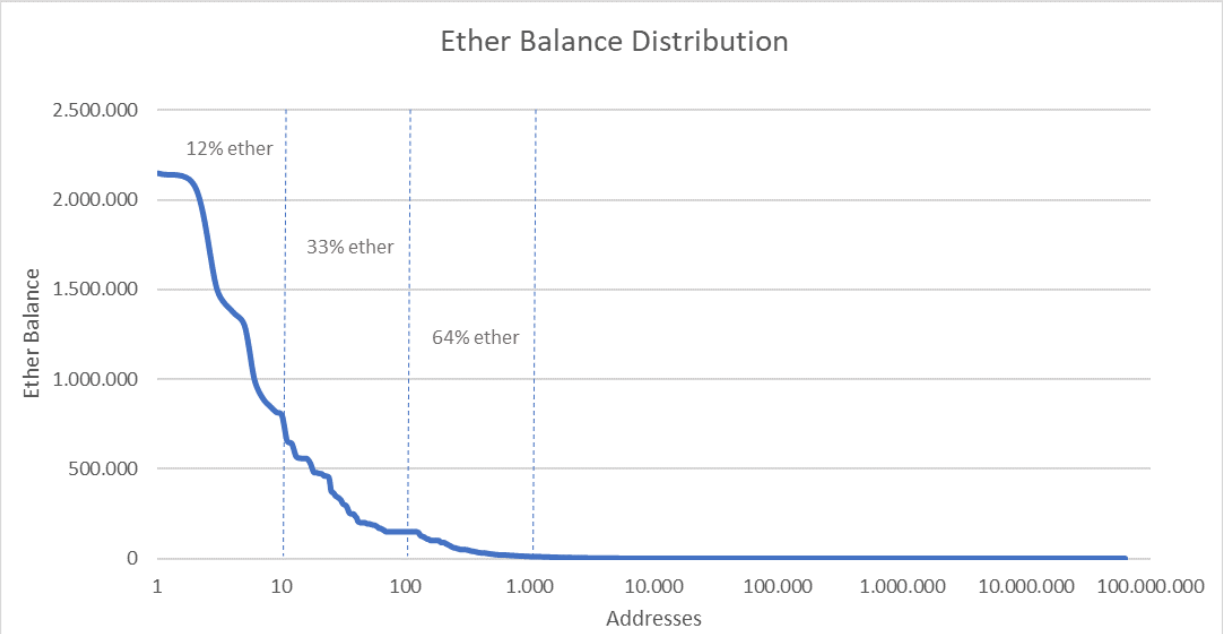


Figure 4.17: Ether distribution among the total number of addresses with an ether balance larger than zero. There is no reason to take into consideration the whole universe of addresses as only the ether balance distribution is analyzed. Both axes are in logarithmic scale.

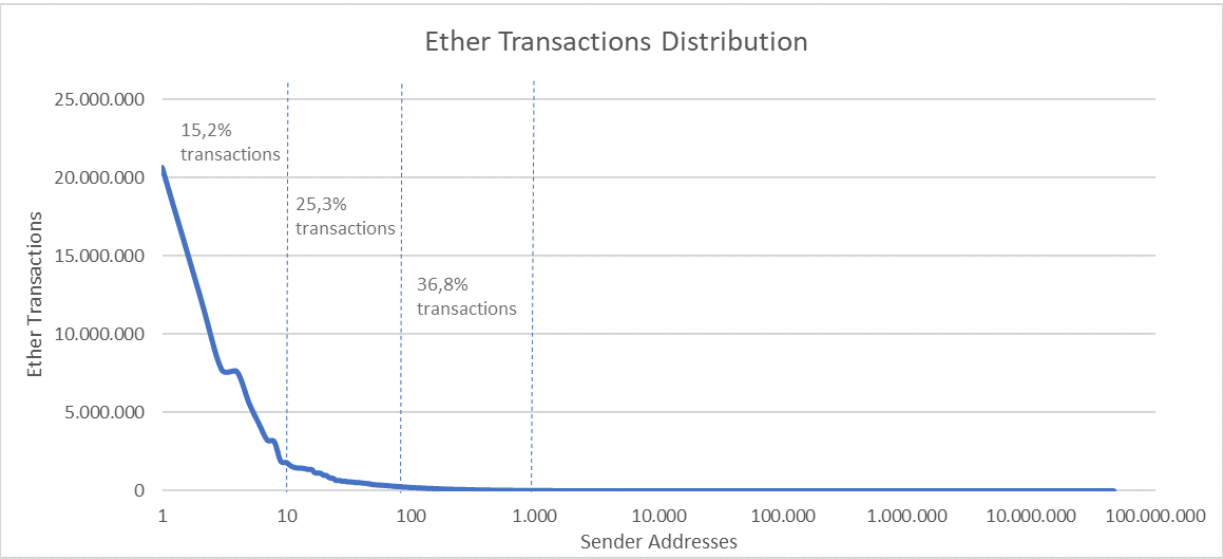


Figure 4.18: Transactions distribution among the total number of addresses that have executed at least one transaction. Both axes are in logarithmic scale.

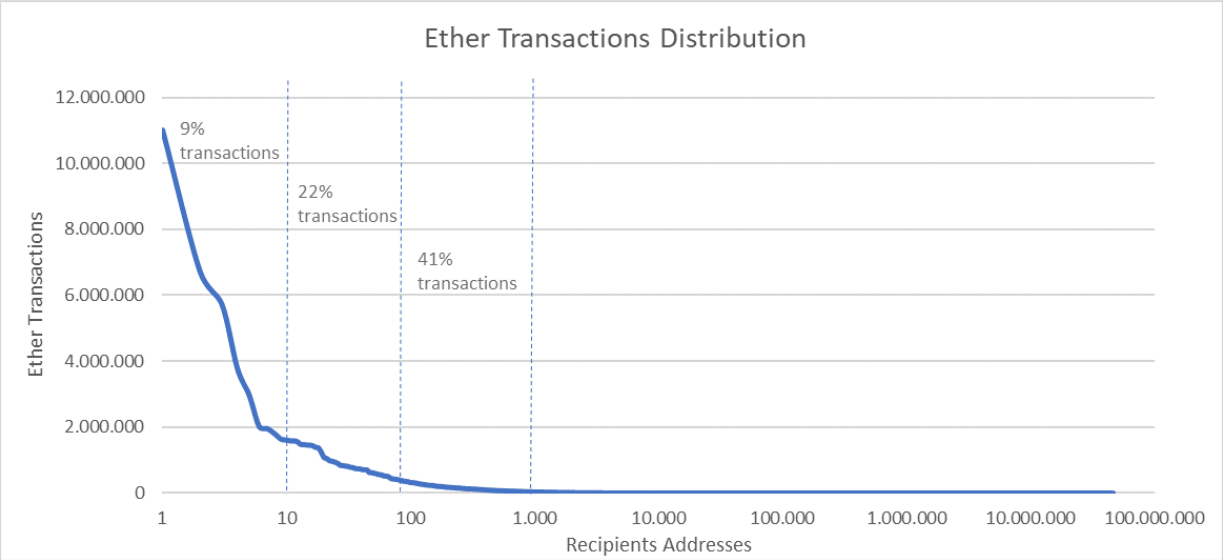


Figure 4.19: Transactions distribution among the total number of addresses that have received at least one transaction. Both axes are in logarithmic scale.

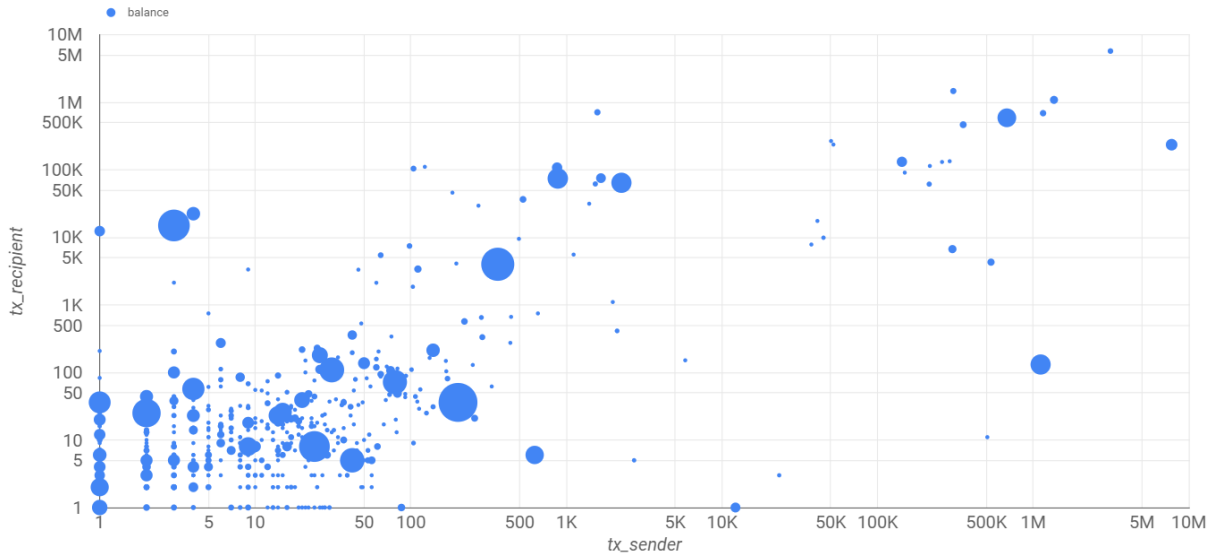


Figure 4.20: Balance and transaction involvement of the top 1'000 addresses. The size of the bubble shows the volume in ether balance. The x axis represents the number of transactions in which the account was the sender and the y axis represents the number of the transactions in which the account was the recipient. These three dimensions represent the activity and the holdings of each address and they are the most valuable data. Both axes are in logarithmic scale.

80%-20%[2], since the 6k addresses are only the 0.025% of the total address population with a strictly positive balance. This concentration of ether among a small number of addresses makes these specific addresses even more important for the whole network.

As it was referred earlier, the Ethereum network is defined by both the number of transactions and the volume of ether. So, in fig. 4.18, the distribution of the number of transactions among the different addresses that have executed transactions as senders is displayed.

It is obvious that the distribution follows once again the pareto rule. The 15,2% of the total transactions are sent from just 10 addresses, out of the approximate 46M addresses that have executed at least one transaction, while the 80% of the total transactions are sent from 7.34% of the addresses.

The distribution of the transactions among the recipients of transactions is a bit different from the one among the senders of the transactions. As shown in fig. 4.19, the top 10 recipients receive 9% of the transactions, a bit lower than the percentage of transactions that are sent by the top 10 sender addresses. At the same time, the 41% of the transactions, almost half of them, are addressed only to 1'000 of the existing addresses in the Ethereum network.

To sum up, only 1'000 addresses play an important role for the circulation of ether in the network. By plotting the data of the top 1'000 addresses by balance in fig. 4.20, it can be analyzed how often these addresses participate in transactions, either as a recipient or as a sender. The size of the bubbles represent the balance of the addresses. The x axis shows the number of transactions in which an address has been the sender, while the y axis shows the number of transactions in which an address has been the recipient.

Hence, each point in fig. 4.20 shows the activity of an account, that is how many transactions a unique address has initiated and in how many transactions the same address has been the recipient. At the same time, the bubble shows the holdings of the account[3].

There are two main conclusions that can be derived out of these plottings:

- First, most of the accounts that hold a valuable amount of ether do not execute so many transactions, which means that they are not very active. Many of these balance

[2]According to the Pareto rule, one would expect that 80% of the ether is held by the 20% of the accounts
[3]The bigger the bubble, the larger the balance of this address.

holders have approximately executed less than 100 transactions and have been the recipients again in less than 100 transactions.

- Second, focusing on the trend of the graph, there is a small linear analogy between the sender and recipient activity of an address. It can be assumed that the ones not following this behaviour represent a specific category. For example, addresses that send and receive many transactions may represent exchange platforms.

Emphasizing in the top 20 addresses, measured by ether balance, lets us identify in which category they belong to. There are some basic categories that have been defined in [Etherscan.io], where all the transactions that are executed in Ethereum network are displayed. The main categories that are used here are exchanges, decentralized exchanges, miners, token contracts and ICO wallets. All the remaining addresses are assumed to be either simple contracts executing specific code or still unlabeled or external addresses. [Etherscan.io] identifies if an address is a contract address or a simple external address. There some clear distinctions among the different categories.

- **Exchanges:** A list of centralized cryptocurrency exchanges which are online platforms that allow customers to buy and sell cryptocurrencies for other assets
- **Decentralized exchanges:** A decentralized exchange (also known as a DEX) is an exchange market that does not rely on a $3^{rd}$ party service to hold the client funds, but, instead, trades occur directly between users (peer to peer) through an automated process [eth]
- **Miners:** Accounts that belong to big mining organizations or single individuals
- **Token contracts:** A Token Contract is a specific type of smart contract that describes (using the contract code) rules of how the token could be generated and transferred between addresses, if it is splittable/ fungible, etc. [Rosic, 2017]

Table 4.3 shows the balance and activity of the top 20 addresses, measured by balance. Given the mentioned categories, the types of these address are assessed. Most of the addresses with the highest ether balance represent exchanges and external addresses.

In table 4.4, the top 10 addresses with the greatest balance of ether are shown, while it is also assessed if they are active or not. An address is defined as active if it has been successfully used during the past 30 days, either sending or receiving a transaction.

All the above ranking of addresses is based on the ether balance. As it was referred earlier, this is only one angle of understanding which address is important for the Ethereum network. The number of transactions is another important indicator. Transactions are defined by an address which sends the transaction and an address which receives the transaction.

In the case of the sender, this can only be an external account (single user, miner, or exchange). A recipient address can be any address (external or contract addresses).

Table 4.3: Top 20 addresses, measured by ether balance (complete addresses can be found in Appendix)

| **Address** | **Balance** | **Nr of received transactions** | **Nr of sent transactions** | **Type** |
|---|---|---|---|---|
| 0xc0...cc2 | 2'152'081 | 362'194 | - | Token contract |
| 0x4e...a67 | 2'074'272 | 36 | 201 | Exchange |
| 0x74...44e | 1'502'074 | 3'994 | 362 | Exchange |
| 0x53...a3d | 1'378'754 | 14'993 | 3 | Exchange |
| 0x66...054 | 1'300'002 | 8 | 24 | Exchange |
| 0xab...83e | 999'999 | 429 | - | Contract |
| 0x61...eea | 895'999 | 152 | - | Exchange |
| 0xdc...0d3 | 850'861 | 109 | 31 | Exchange |
| 0xfc...8e4 | 817'000 | 5 | 42 | External address |
| 0xe8...919 | 801'053 | 72 | 79 | Exchange |
| 0x22...0f6 | 657'334 | 36 | 1 | External address |
| 0xde...bae | 642'538 | 519 | - | Contract |
| 0x13...9c5 | 570'981 | 74'411 | 880 | External address |
| 0x1b...7c2 | 560'000 | 97 | - | External address |
| 0x7e...164 | 558'164 | 64'224 | 2'257 | External address |
| 0x26...dc0 | 556'489 | 131 | 1'116'688 | Exchange |
| 0x51...0eb | 530'000 | 101 | - | External address |
| 0x84...0ed | 483'000 | 23 | 14 | External address |
| 0x6c...a7b | 479'393 | 589'011 | 677'395 | Exchange |
| 0x74...e82 | 475'000 | 8 | 9 | External address |

In table 4.5, the top 20 addresses, measured by the number of transactions they send, are presented. Earlier in this chapter, after plotting the transactions distribution in the different addresses (see fig. 4.18), it was assumed that most of them should be exchanges given that this is the main operation of the exchanges. Indeed, based on the data, most of

Table 4.4: Top 10 addresses, measured by ether balance

| Address | Name | Active |
|---|---|---|
| 0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2 | Wrapped Ether | + |
| 0x4e9ce36e442e55ecd9025b9a6e0d88485d628a67 | Binance 6 | + |
| 0x742d35cc6634c0532925a3b844bc454e4438f44e | Bitfinex 5 | + |
| 0x53d284357ec70ce289d6d64134dfac8e511c8a3d | Kraken 6 | − |
| 0x66f820a414680b5bcda5eeca5dea238543f42054 | Bittrex 3 | + |
| 0xab7c74abc0c4d48d1bdad5dcb26153fc8780f83e | NaN | + |
| 0x61edcdf5bb737adffe5043706e7c5bb1f1a56eea | Gemini 3 | + |
| 0xdc76cd25977e0a5ae17155770273ad58648900d3 | Huobi 6 | − |
| 0xfca70e67b3f93f679992cd36323eeb5a5370c8e4 | NaN | + |
| 0xe853c56864a2ebe4576a807d26fdc4a0ada51919 | Kraken 3 | − |

the exchanges that represent some of the larger in ether balance accounts are also in the top 20 senders of transactions. However, miners win this battle by finding themselves in almost all the 5 top places, even if their balances are not as high. This is totally logical, given that miners validate all the available transactions in the network and they do not invest in ether. The balance created is from the rewards they receive during the mining process.

At the same time, it is important to also identify the top addresses, measured by the number of transactions they have received. In table 4.6, these addresses are displayed. This is a more diversified sample of addresses, since one can find exchanges, token contracts, as well as general contracts. In the $6^{th}$ place, there is an empty address. Recalling the references on contract creation, this row represents all the transactions that have been executed in order to create a new contract. This is the reason that the field "to_address" is empty.

It is impressing that although there are exchanges and token contracts in the list, these are not the same with the ones in table 4.5. Moreover, there are a lot of contracts that are recipients of transactions but they do not belong to the token contracts category. There are two possible explanations: either they are unclassified in [Etherscan.io], which would be a bit strange given that they belong to the top 20 of the recipients, or they are execution contracts that execute some orders.

Table 4.5: Top 20 addresses, measured by number of transactions they have sent (complete addresses can be found in Appendix)

| Address | Nr of sent transactions | Balance | Name | Type |
|---|---|---|---|---|
| 0xea...ec8 | 20'629'170 | 411 | Ethermine | Miner |
| 0x52...3b5 | 12'552'279 | 6'604 | Nanopool | Miner |
| 0xfb...b98 | 7'761'415 | 183'525 | Bittrex 1 | Exchange |
| 0x82...830 | 7'617'805 | 2'075 | F2Pool 2 | Miner |
| 0x5a...c4c | 5'617'075 | 4'642 | Spark Pool | Miner |
| 0x2a...226 | 4'303'185 | 833 | DwarfPool 1 | Miner |
| 0xa7...49e | 3'252'972 | 25 | IDEX 2 | Decentralized exchange |
| 0x3f...0b3 | 3'139'151 | 42'195 | Binance 1 | Exchange |
| 0xce...a32 | 1'908'247 | - | NaN | External address |
| 0xb2...347 | 1'811'672 | 8'764 | MiningPoolHub | Miner |
| 0x6c...21f | 1'572'127 | 199 | NaN | External address |
| 0x2b...258 | 1'471'477 | 8'136 | KuCoin 1 | Exchange |
| 0xd5...2ff | 1'458'459 | 6'950 | Binance 2 | Exchange |
| 0x06...bbf | 1'432'310 | 7'456 | Binance 4 | Exchange |
| 0x56...ced | 1'367'834 | 5'791 | Binance 3 | Exchange |
| 0x32...d88 | 1'361'217 | 87'538 | Poloniex 1 | Exchange |
| 0x0d...2fe | 1'157'681 | 54'716 | Gate.io 1 | Exchange |
| 0xd3...375 | 1'150'417 | - | Genesis Mining | Miner |
| 0x26...dc0 | 1'116'688 | 556'489 | Kraken 4 | Exchange |
| 0x61...bd9 | 994'929 | - | F2Pool 1 | Miner |

Moreover, a new category has appeared, which is ENS. According to [Etherscan.io], ENS (Ethereum Name Service) offers a secure & decentralised way to address resources both on and off the blockchain using simple, human-readable names.

Again, the accounts that receive most of the transactions in the network just own some very few ether. This makes sense as most of them are contracts and they are not supposed

Table 4.6: Top 20 addresses, measured by number of transactions they have received (complete addresses can be found in Appendix)

| Address | Nr of received transactions | Balance | Name | Type |
|---|---|---|---|---|
| 0x8d...819 | 10'998'207 | 23'657 | EtherDelta 2 | Decentralized exchange |
| 0x2a...208 | 6'721'660 | 45'095 | IDEX 1 | Decentralized exchange |
| 0x3f...0be | 5'731'059 | 42'195 | Binance 1 | Exchange |
| 0x06...66d | 3'790'435 | 60 | CryptoKitties: Core | Token contract |
| 0x86...db0 | 2'959'558 | - | EOS | Token contract |
| - | 2'609'558 | - | NaN | Contract creation process |
| 0xf2...2e2 | 2'023'290 | - | Tron Token | Token contract |
| 0x70...413 | 1'954'539 | 4'461 | ShapeShift 3 | Exchange |
| 0x20...8ef | 1'800'667 | - | Poloniex 2 | Exchange |
| 0xd1...405 | 1'639'262 | 209 | Dice2Win | Contract |
| 0xe9...ff3 | 1'609'053 | - | Bittrex 2 | Exchange |
| 0xa3...770 | 1'578'576 | - | NaN | Contract |
| 0xd2...c07 | 1'573'657 | - | OmiseGO Token | Token contract |
| 0x60...76b | 1'479'089 | 98 | ZB.com | Exchange |
| 0xf5...4f3 | 1'466'737 | 51'183 | Yobit.net | Exchange |
| 0x03...b0e | 1'452'736 | - | NaN | Contract |
| 0xfa...0b3 | 1'447'475 | - | Kraken 1 | Exchange |
| 0x60...8ef | 1'393'672 | - | NaN | ENS |
| 0x0e...50f | 1'373'336 | - | NaN | Contract |
| 0x17...ab9 | 1'242'360 | - | NaN | Contract |

to hold ether but only to execute a sequence of orders.

Finally, there are three token contracts with a very low ether balance. These are the cryptokitties, EOS, and TRON tokens. CryptoKitties is one project utilizing ERC-721 tokens, which focuses on gaming where users can collect virtual cats. Each cat is represented

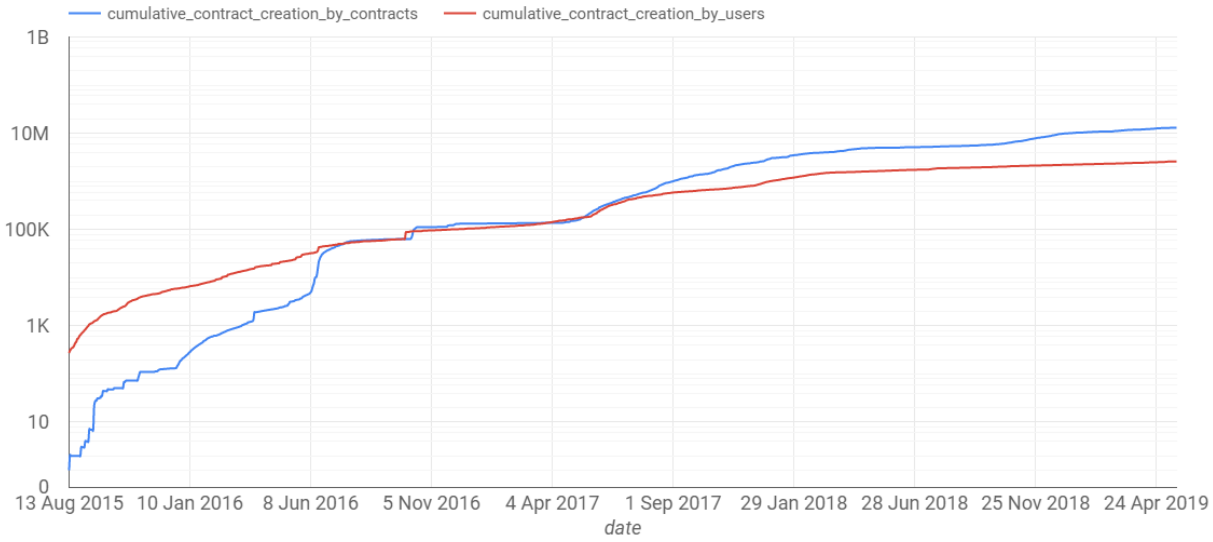


Figure 4.21: Cumulative number of contracts created by users and of contracts created by other contracts (logarithmic scale)

by an Ethereum ERC-721 token, which means that they are all one-of-a-kind and can never be replicated, taken away from the owner, or destroyed [Lai and O'Day, 2018]. EOS and TRON are older ICOs. The reason they have no balance is because they have been used as ICO vehicles in order to receive some funding and then the funds were transferred to the beneficiary account, according to the contract format.

## 4.6 Analysis of smart contracts

Regarding only the contract addresses, as it is referred in section 4.2.6, there are in total 15'571'524 contracts. For these contracts, the source of creation is analyzed. There are contracts, which are created directly from an external address and contracts, which are contract-created. In the former ones, the user has to set recipient address (to_address) to zero. In the latter ones, an external user calls the "create" function of contract A, which in turn creates contract B. This is a message call (internal transaction) and it is not counted in the number of transactions. As a result, only the direct contract creations are considered transactions and represent only 1% of the total transactions. The indirect contract creations are reflected in the contract calls.
It is very interesting to see in fig. 4.21 that most of the contracts were initially user-created contracts. However, as Ethereum is getting some notice and developers start participating and building more complicated contracts, contract-created contracts dominate the first ones since mid-2017. In fact, today there are over 13M contact-created contracts, while the user-created contracts are only over 2M. Looking into the data, most of the user-created contracts are Ethereum-based currencies that use contract-created contracts for some functionalities. This means that an important fraction of the activity in the Ethereum network

is using the more complicated features of the contracts, such as the creation of another contract.

Diving into the contracts, we want to identify who is creating these contracts and how many of these contracts are 100% identical. In order to identify the identical contracts, the output of the traces will be compared. The output is the bytecode of the contract when a new contract is created. The bytecode is the hex representation of what a contract is created to execute.

Table 4.7: Different types of contracts

| **Type of contract** | **Nr of contracts** | **Nr of unique creators** | **Nr of unique bytecodes** |
|---|---|---|---|
| User-created contracts | 2'609'558 | 105'444 | 202'516 |
| Contract-created contracts | 13'056'526 | 32'142 | 7'288 |

From the data in table 4.7, it is concluded that a very small number of external users have directly created contracts. What is even more impressing is that only 32K contracts are responsible for the creation of 13M contracts.

Comparing the bytecode of the user-created contracts, some of them are identical, as there are only 200K different bytecodes in the total of 2M contracts. At the same time, comparing the bytecode of the contract-created contracts, only 7K of unique bytecodes are recycled in a total of 13M contracts. Summing up the two types of contracts, it is concluded that only 1.34% of the total contracts are completely unique.

# Chapter 5

# Smart contracts and finance industry

Blockchain technology and especially smart contracts have been considered a revolutionary advancement that can shape the finance industry of tomorrow, and specifically the banking industry. On the other side, banks have been reluctant to use the current permissionless environments because of the lack of privacy in the current setups.

During the past years, many big banks have been involved in consortiums assessing blockchain solutions and implementations. On top of that, Societe Generale recently issued a $112 million bond, using smart contracts built, not on a private, permissioned blockchain, but on the public, permissionless ethereum blockchain [Casey, 2019]. This means that the whole industry is getting more open and does not anymore see blockchain as a risk, but as a technology revolution that should at least be followed and understood.

Ethereum introduced the idea of smart contracts almost five years ago. Especially in the banking industry, smart contracts could change the whole landscape in some major areas:

1. Smart contracts can replace the time consuming, as well as complicated legal documentation processes. All this data will be part of a totally transparent and accessible system for regulators.

2. One of the best candidate processes that can change dramatically with the use of smart contracts is the process of clearing and settlement. The entire lifecycle of a trade – execution, clearing, and settlement can be greatly facilitated, while the counterparty risk can be lowered.

3. Blockchain technology can make it possible for banks to create a direct communication link between each other, leading to frictionless cross border transactions. Corda is working on a case of having two banks sharing the same ledger for transactions and contracts [Bauerle, 2019].

4. In the sharing economy, anything in the future is expected to be investable. Not only the traditional asset classes, such as equities and bonds, but also physical assets, such as art and real estate can be offered directly to the clients through smart contracts.

5. The concept of digital identity can enable banks to efficiently store and update client information.

The technology is already in place. What is the most important part is the construction of the right contracts that will not leave space for vulnerabilities and mis-implementations.

# Chapter 6

# Conclusion

## 6.1 Main findings

Summarizing all the research that has been done in the area of the Ethereum network, the following conclusions can be made.

1. The number of transactions and the value of ether transferred are two good indicators to understand how the network behaves. Comparing these two, we have seen that they have not followed the same path during the last 4 years. The number of transactions has an exponential growth, while the value of transferred ether fluctuates a lot, day to day. However, we have identified, for both of them, a stabilization during the last year at around 700'000 transactions and 2'000'000 ETH per day, respectively.

2. At the same, looking at the ether price over this period, we have concluded that it follows the pattern of the number of transactions. This fact helped us reach the conclusion that the number of transactions may affect the price of ether, or vice versa. We did not dive into this causation relationship. However, we could say that this could be an explanation of the ether price speculation.

3. Although the growth rate of the number of transactions has dropped dramatically in the middle of 2018, the gas used was continuously increasing, even today. This is the first indicator that there have been more contracts in the network, taking place and requiring even more gas than a normal ether transfer.

4. We have verified that there are approximately 60M addresses in the Ethereum network as of $20^{th}$ of May 2019. This number includes the external addresses, miner addresses and all the contracts that represent almost 37% of the total address population. However, it excludes the random addresses that have been created from attacks, such as DDOS.

5. Focusing on the transactions, we have found out that almost 45% of them are contract calls and 54% of them are simple ether transfers. Only 1% is the contract creation

ones. Looking how these numbers have been formatted during the last year, we have identified a huge move from ether transfer activity to contract call activity.

6. This 1% of the transactions represents the direct action of a user to create a contract. However, we have seen that most of the existing contracts have been created from other contracts. More precisely, there are approximately 13M contract-created contracts, out of the universe of 15M contracts. This is another fact that indicates that the Ethereum network has become more contract-focused than transaction-focused.

7. Diving into the contracts, there is one true fact: there is a lack of diversity in the smart contracts ecosystem. There are few creators of the contracts, relatively to their number, for both types of contracts (user-created and contract-created contracts). Moreover, this lack of diversity exists even in the bytecode. This means that most of the contracts in the Ethereum network are identical. More particularly, only 1.34% of the total contracts are completely unique. It remains to see how this similarity and dependency may affect the vulnerability of the whole network.

8. Doing a top down analysis focusing on distinct addresses, we have found out that 12% of the total ether is held by the top 10 addresses, measured by balance, and 80% is held by 6K addresses in total. This outcome shows in practice the 80%-20% pareto rule and that only few addresses are very important for the whole network. This could be a second reason for the price speculation. However, any causation is not tested in this thesis.

9. A similar behavior exists in the transactions distribution where 41% of the transactions are addressed to only 1'000 addresses and 36.8% of the transactions are sent from only 1'000 addresses, respectively. For both the recipients and the senders of transactions, there is a long tail that represents the whole population of almost inactive addresses.

10. In the end, we have focused on the analysis of specific addresses, taking into consideration their holdings and the number of transactions (both as senders and recipients). Most of the accounts that hold a valuable amount of ether do not execute so many transactions (just few hundreds). Normally, addresses with high ether balance are either exchanges or external users (individuals). However, this would be a bit strange for exchanges, since they normally execute a huge number of transactions. This is something that needs further exploration, or even clustering of the different exchanges that present a different behavior. On the other side, taking into consideration the top 20 addresses that execute the most transactions, we have seen that exchanges are in the top of the list, as it was expected. However, miners have a dominant position in that list, as well. Some unlabeled contracts appear in the list, which means that they do not belong to a specific category, but they are mostly used as executional vehicles. The main conclusion out of this is, in any case, that, in most of the cases, the number of transactions in which an address is involved is inversely proportional to the ether holdings of this account.

## 6.2 Related work

Because of the exponential growth of the digital currencies after the Bitcoin release and establishment, there is extensive work done in order to understand the digital currencies in total, and more particularly Ethereum, as it is the second in market cap after Bitcoin. However, there are not many textbooks focusing on Ethereum. Some of the information comes from published papers. However, most of the content comes from blogs, webpages that focus on digital currencies, github[1], and many technology articles. The best source of truth regarding a digital currency is always the white paper published for each that specific currency.

Early work in this area has been done in [Int, 2018], analyzing the Ethereum's contract topology, and more specifically analyzing how contracts are being used. This research is more an attempt to cluster contacts, given the similarity of the bytecode. Moreover, M. De Aliaga in [Aliaga, 2018] tried as well to classify the different addresses that exist in Ethereum, although covering only a few number of labeled data. Furthermore, R. Norvill et al. published [R. Norvill and R. State and I. Awan and B. B. F. Pontiveros and A. Cullen, 2017], which goes one step further in the classifying process, using a k-means approach, and could be considered future work. Finally, N. He et al. has tried in [He et al.] to capture the vulnerability that may exist in the different contracts because of their similarity.

At the moment, there are also many supportive platforms, such as etherscan, which monitors all the Ethereum activity, and Google Big query, which stores all the data in an SQL database and is the one used for this research.

## 6.3 Future exploratory paths

Digital currencies and decentralized networks are the future in many industries. They will be regarded as the internet of today; accessible to every business to use, providing at the same time the transparency of data that internet sometimes misses. So, the ground for future analysis is limitless:

1. In this thesis, the labels of the different addresses have been extracted from [Etherscan.io]. More particularly, the top 100 exchanges, decentralized exchanges, miners, ICO wallets and tokens are extracted. However, there are still many contracts and external addresses that have not yet be labeled. It would be useful to use a machine learning algorithm that can label any address according to the behavior of the address or the bytecode in case of contract. This could be even an improvement for the

[1]The world's leading development platform where people upload their code or ask related questions

Ethereum network. Most of the labeling today in the research papers is done either manually or using the etherscan platform.

2. In this thesis, the main analysis is concentrated around addresses that are always measured by ether value (balance) and number of transactions (traffic). More precisely, only transactions that have been initialized from external accounts are considered, as this explicitly occurs in any transaction. As a future work, tt would be interesting to focus only on the message calls among contracts and to find out what the added value of having chains of contract calls is to the final output.

3. On $18^{th}$ of May 2019, a new hard fork, Casper, occurred in the Ethereum network. The fork brought a change to the consensus mechanism from proof of work to Casper. Casper is a partial consensus mechanism combining the proof of stake and the Byzantine fault tolerant consensus [V. Buterin and V. Griffith]. This combination works more as a hybrid system, having the two consensus mechanisms working in parallel. However, the goal is to pivot totally in the proof of stake consensus. Some of the hypothetical outcomes are the reduced electricity usage, less centralization of the mining process, reduced risk of attack, scalability of transactions execution frequency, limitation of issuing as many new coins [Falk, 2019]. All these potential outcomes will change totally the way Ethereum operates and its evaluation, so an analysis of the actual outcomes would be interesting.

4. On $18^{th}$ of June 2019, Facebook announced a new digital currency, Libra [Silva, 2019]. Libra has managed to concentrate all the advantages of all other digital currencies that have been tested for years and learn from all these trials. It will not be Facebook alone on this huge project, but a consortium of companies, the so called Libra Association, specializing in a number of related industries, such as the financial, the e-commerce, the tech and the telecommunications industry. As a result, given some of the many advantages it has, such as an established user database (almost 2.7 billion customers in the platform), the support of companies like Visa, the consensus and the pegging to a basket of sovereign-issued currencies [Reiff, 2019], it would be very interesting to compare Libra to existing currencies and analyze how it will affect the future of digital currencies.

# Bibliography


Etherscan accounts dex. https://etherscan.io/accounts/label/dex. Accessed: 2019-07-01.

Ł. Żuchowski. Ethereum: Everything you want to know about gas. https://blog.softwaremill.com/ethereum-everything-you-want-to-know-about-the-gas-b7c8f5c17e7c, 2017. Accessed: 2019-05-26.

A. Lewis. A gentle introduction to ethereum. https://bitsonblocks.net/2016/10/02/gentle-introduction-ethereum/, 2016. Accessed: 2019-05-23.

A. Sassano. Why ether is valuable. https://medium.com/ethhub/why-ether-is-valuable-2b4e39e01eb3, 2019. Accessed: 2019-05-23.

A. Sokolowska. How to interact with the ethereum blockchain and create a database with python and sql. https://medium.com/validitylabs/how-to-interact-with-the-ethereum-blockchain-and-create-a-database-with-python-and- 2018. Accessed: 2019-05-23.

M. De Aliaga. Classifying ethereum users using blockchain data. https://medium.com/tokenanalyst/classifying-ethereum-users-using-blockchain-data-dd6edb867de3, 2018. Accessed: 2019-07-02.

D. Appelbaum and S. S. Smith. Blockchain basics and hands-on guidanc. *The CPA Journal*, 2018.

B. Asolo. Blockchain public key and private key: A detailed guide. https://www.mycryptopedia.com/public-key-private-key-explained/, 2019. Accessed: 2019-05-11.

B. Curran. What is practical byzantine fault tolerance? complete beginner's guide. https://blockonomi.com/practical-byzantine-fault-tolerance/, 2018. Accessed: 2019-05-11.

N. Bauerle. How could blockchain technology change finance? https://www.coindesk.com/information/how-blockchain-technology-change-finance, 2019. Accessed: 2019-07-02.

BBVA. What's the difference between dlt and blockchain? https://www.bbva.com/en/difference-dlt-blockchain/, 2018. Accessed: 2019-05-11.

Blockgeeks. Proof of work vs. proof of stake. https://blockgeeks.com/guides/proof-of-work-vs-proof-of-stake/, a. Accessed: 2019-05-11.

Blockgeeks. What is hyperledger? the most comprehensive guide ever! https://blockgeeks.com/guides/hyperledger/, b. Accessed: 2019-05-11.

Blocktonite. Private vs. public and permissioned vs. permission-less. https://blocktonite.com/2017/06/27/private-vs-public-and-permissioned-vs-permission-less/, 2017. Accessed: 2019-05-11.

M. J. Casey. A glimpse of banking's future, live on the ethereum blockchain. https://www.coindesk.com/societe-generales-work-with-public-ethereum-is-a-big-deal, 2019. Accessed: 2019-07-02.

Coinmama. History of ethereum. https://www.coinmama.com/guide/history-of-ethereum. Accessed: 2019-05-23.

CoinMarketCap. https://coinmarketcap.com/. Accessed: 2019-05-05.

corda.net. The ledger. https://docs.corda.net/key-concepts-ledger.html. Accessed: 2019-05-11.

D. Brickwood. Understanding trie databases in ethereum. https://medium.com/shyft-network-media/understanding-trie-databases-in-ethereum-9f03d2c3325d, 2018. Accessed: 2019-05-23.

D. Hao. Ethereum: What it is, why it's important, and how it's building tomorrow. https://medium.com/the-ledger-group/ethereum-what-it-is-why-its-important-and-how-it-s-building-tomorrow-278933c18b4b, 2018. Accessed: 2019-05-05.

D. Vujicic and D. Jagodic and S. Randic. Blockchain technology, bitcoin, and ethereum: A brief overview. In *Blockchain technology, bitcoin, and Ethereum: A brief overview*. IEEE, 2018.

A. Day and E. Medvedev. Ethereum in bigquery: a public dataset for smart contract analytics. https://cloud.google.com/blog/products/data-analytics/ethereum-bigquery-public-dataset-smart-contract-analytics, 2018. Accessed: 2019-06-05.

E. Medvedev and A. Day. Ethereum in bigquery: how we built this dataset. https://cloud.google.com/blog/products/data-analytics/ethereum-bigquery-how-we-built-dataset, 2018. Accessed: 2019-05-23.

Easy Ethereum. Key milestones for ethereum. https://www.easyeth.com/key-milestones-for-ethereum.html. Accessed: 2019-05-23.

ETH events. Why reading data from the ethereum blockchain is hard and how to speed it up. https://eth.events/news/why-reading-data-from-the-ethereum-blockchain-is-hard-and-how-to-speed-it-up/, 2018. Accessed: 2019-05-23.

ETH Gas Station. https://ethgasstation.info/. Accessed: 2019-05-26.

Etherscan.io. Etherscan.io. https://etherscan.io/. Accessed: 2019-06-03.

F. Gadaleta. This is how ethereum works. https://medium.com/fitchain/this-is-how-ethereum-works-60f37abd5ef5, 2018. Accessed: 2019-05-23.

T. Falk. Ethereum's casper protocol explained in simple terms. https://www.finder.com/ethereum-casper, 2019. Accessed: 2019-07-02.

G. Konstantopoulos. Proof of work vs. proof of stake. https://medium.com/loom-network/understanding-blockchain-fundamentals-part-2-proof-of-work-proof-of-st 2017. Accessed: 2019-05-11.

G. Wood. Ethereum yellow paper - ethereum: A secure decentralised generalised transaction ledger. Technical report, ethereum.org.

H. Anwar. The ultimate comparison of different types of distributed ledgers: Blockchain vs hashgraph vs dag vs holochain. https://101blockchains.com/blockchain-vs-hashgraph-vs-dag-vs-holochain/, 2018. Accessed: 2019-05-11.

H. Kenneth. Ethereum account. https://medium.com/coinmonks/ethereum-account-212feb9c4154, 2018. Accessed: 2019-05-26.

N. He, L. Wu, H. Wang, Y. Guo, and X. Jiang. Characterizing code clones in the ethereum smart contract ecosystem.

*Analyzing Ethereum's Contract Topology*, 2018. Internet Measurement Conference.

J. Kelly. Nine of world's biggest banks join to form blockchain partnership. https://www.reuters.com/article/us-banks-blockchain/nine-of-worlds-biggest-banks-join-to-form-blockchain-partnership-idUSKCN0RF24M2015 2015. Accessed: 2019-05-11.

J. P. Buntinx. The history of ethereum in 500 words. https://themerkle.com/the-history-of-ethereum-in-500-words/, 2017. Accessed: 2019-05-23.

K. Kim. Modified merkle patricia trie - how ethereum saves a state. https://medium.com/codechain/modified-merkle-patricia-trie-how-ethereum-saves-a-state-e6d7555078dd, 2018. Accessed: 2019-05-23.

Katalyse.io. Blockchain basics - what is evm. https://cryptodigestnews.com/blockchain-basics-what-is-evm-52d83616764, 2018. Accessed: 2019-05-23.

V. Lai and K. O'Day. Ethereum erc token standards. https://crushcrypto.com/ethereum-erc-token-standards/, 2018. Accessed: 2019-07-01.

Lisk.io. Delegated proof of stake. https://lisk.io/academy/blockchain-basics/how-does-blockchain-work/delegated-proof-of-stake. Accessed: 2019-05-11.

M. Beedham. Here's the difference between 'permissioned' and 'permissionless' blockchains. https://thenextweb.com/hardfork/2018/11/05/permissioned-permissionless-blockchains/, 2018. Accessed: 2019-05-11.

M. Thake. What is dag distributed ledger technology? https://medium.com/nakamo-to/what-is-dag-distributed-ledger-technology-8b182a858e19, 2018a. Accessed: 2019-05-11.

M. Thake. What's the difference between blockchain and dlt? https://medium.com/nakamo-to/whats-the-difference-between-blockchain-and-dlt-e4b9312c75dd, 2018b. Accessed: 2019-05-11.

D. Mingxiao, M. Xiaofeng, Z. Zhe, W. Xiangwei, and C. Qijun. A review on consensus algorithm of blockchain. *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2017.

N. Avramov. How r3 corda works. https://medium.com/@nickavramov/how-r3-corda-works-24d9285059a2, 2019. Accessed: 2019-05-11.

H. Natarajan, S. K. Krause, and H. L. Gradstein. Distributed ledger technology (dlt) and blockchain. *FinTech Note*, 1, 2017.

O. Belin. The difference between blockchain and distributed ledger technology. https://tradeix.com/distributed-ledger-technology/. Accessed: 2019-05-11.

R. G. Brown and J. Carlyle and I. Grigg and M. Hearn. Corda white paper - corda: An introduction. Technical report, corda.net.

R. Norvill and R. State and I. Awan and B. B. F. Pontiveros and A. Cullen. Automated labeling of unknown contracts in ethereum. In *26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2017.

N. Reiff. Facebook gathers companies to back cryptocurrency launch. https://www.investopedia.com/facebook-gathers-companies-to-back-cryptocurrency-launch-4690619, 2019. Accessed: 2019-07-02.

A. Rosic. What is an ethereum token: The ultimate beginner's guide. https://blockgeeks.com/guides/ethereum-token/?fbclid=IwAR1wZJtMF4LnKE6b4b8e5KrEmhlY4lmToO4Kz8r4rzyRIGo3KI2mTI1H6Yc, 2017. Accessed: 2019-07-01.

S. Ray. The difference between blockchains and distributed ledger technology. https://towardsdatascience.com/the-difference-between-blockchains-distributed-ledger-technology-42715a0fa92, 2018a. Accessed: 2019-05-05.

S. Ray. What is a hashgraph? https://hackernoon.com/what-is-a-hashgraph-a0b4c7c396d2, 2018b. Accessed: 2019-05-11.

R. Shuwar and O. Vashchuk. Pros and cons of consensus algorithm proof of stake. difference in the network safety in proof of work and proof of stake. *Electronics and information technologies*, 9:106–112, 2018.

C. Sillaber and B. Waltl. Life cycle of smart contracts in blockchain ecosystems. *Datenschutz und Datensicherheit - DuD*, 41:497–500, 2017.

M. De Silva. The winners and losers of facebook's libra. https://qz.com/1655319/the-winners-and-losers-of-facebooks-libra/, 2019. Accessed: 2019-07-02.

Solidity. Solidity. https://solidity.readthedocs.io/en/v0.5.8/, 2019. Accessed: 2019-05-23.

T. McCallum. Diving into ethereum's world state. https://medium.com/cybermiles/diving-into-ethereums-world-state-c893102030ed, 2018. Accessed: 2019-05-23.

T. Schumann. Consensus mechanisms explained. https://hackernoon.com/consensus-mechanisms-explained-pow-vs-pos-89951c66ae10, 2018. Accessed: 2019-05-05.

Upfolio. Ethereum explained. https://www.upfolio.com/ultimate-ethereum-guide. Accessed: 2019-05-11.

V. Buterin. Ethereum white paper - a next generation smart contract and decentralized application platform. Technical report, ethereum.org.

V. Buterin and V. Griffith. Casper the friendly finality gadget. Technical report, Ethereum Foundation.

V. Lai. Ethereum erc token standards. http://crushcrypto.com/ethereum-erc-token-standards/, 2018. Accessed: 2019-05-23.

M. Valenta and P. Sandner. Comparison of ethereum, hyperledger fabric and corda. *Frankfurt School Blockchain Center*, 2017.

# Part II

# Appendix

**SQL code for all the different queries used in order to derive the proper information. Some of them are a mix of others, as the research questions were every time different**

- Number total transactions as of today
  select count(transactions.hash)
  from '$bigquery - public - data.ethereum\_blockchain.transactions$' as transactions
  where block_timestamp $< '2019 - 05 - 21'$
- Average number of transactions
  select count(*) / count(distinct(date(block_timestamp)))
  from '$bigquery-public-data.ethereum\_blockchain.transactions$' where block_timestamp $< '2019 - 05 - 21'$
- Number of transactions per day
  select date(block_timestamp) as date, count(*) as number_of_transactions
  from '$bigquery-public-data.ethereum\_blockchain.transactions$' where block_timestamp $< '2019 - 05 - 21'$ group by date order by date
- Total value of ether transferred as of today
  select sum(value)/power(10,18)
  from '$bigquery - public - data.ethereum\_blockchain.transactions$'
- Total value of ether transferred
  select date(block_timestamp) as date, sum(value)/power(10,18) as ether_value
  from '$bigquery-public-data.ethereum\_blockchain.transactions$' where block_timestamp $< '2019 - 05 - 21'$ group by date order by date
- Ether supply growth rate
  with a as ( select date(block_timestamp) as date, sum(value) as value
  from '$bigquery-public-data.ethereum\_blockchain.traces$' where trace_type in ($'genesis'$, $'reward'$) group by date(block_timestamp) ) select date, sum(value) OVER (ORDER BY date) / power(10, 18) AS supply from a
- Gas used per day (in blocks)
  select date(timestamp) as date, sum(gas_used) as gas_used
  from '$bigquery-public-data.ethereum\_blockchain.blocks$' where timestamp $< '2019-05 - 21'$ and timestamp $> '2015 - 07 - 19'$ group by date order by date
- Average gas price per day (post Byzantium)
  select date(block_timestamp) as date, avg(gas_price) as average_gas_price
  from '$bigquery-public-data.ethereum\_blockchain.transactions$' where block_timestamp $< '2019 - 05 - 21'$ and block_timestamp $> '2015 - 07 - 19'$ and receipt_status $= 1$ group by date order by date

- Address growth per day
  with value_table as ( select to_address as address, value as value, block_timestamp from ‘*bigquery − public − data.ethereum_blockchain.traces*‘ where to_address is not null and status = 1 and (call_type not in (*′delegatecall′*, *′callcode′*, *′staticcall′*) or call_type is null) union all select from_address as address, -value as value, block_timestamp from ‘*bigquery − public − data.ethereum_blockchain.traces*‘ where from_address is not null and status = 1 and (call_type not in (*′delegatecall′*, *′callcode′*, *′staticcall′*) or call_type is null) union all select miner as address, sum(cast(receipt_gas_used as numeric) ∗ cast(gas_price as numeric)) as value, block_timestamp from ‘*bigquery − public − data.ethereum_blockchain.transactions*‘ as transactions join ‘*bigquery−public−data.ethereum_blockchain.blocks*‘ as blocks on blocks.number = transactions.block_number group by blocks.miner, block_timestamp union all select from_address as address, -(cast(receipt_gas_used as numeric) ∗ cast(gas_price as numeric)) as value, block_timestamp from ‘*bigquery − public − data.ethereum_blockchain.transactions*‘), value_table_by_date as ( select address, sum(value) as balance_increment, date(block_timestamp) as date from value_table group by address, date), daily_balances_with_gaps as ( select address, date, sum(balance_increment) over (partition by address order by date) as balance, lead(date, 1, current_date()) over (partition by address order by date) as next_date from value_table_by_date), calendar as ( select date from unnest(generate_date_array(*′*2015−07−30*′*, current_date())) as date), daily_balances as ( select address, calendar.date, balance from daily_balances_with_gaps join calendar on daily_balances_with_gaps.date <= calendar.date and calendar.date < daily_balances_with_gaps.next_date) select date, count(∗) as address_count from daily_balances where balance > 0 group by date

- Total number of contracts (all contracts)
  select count(distinct(address)) as number_of_contracts from ‘*bigquery−public−data.ethereum_blockchain.contracts*‘ where block_timestamp < *′*2019 − 05 − 21*′*

- Cumulative number of contracts created
  with a as (select date(block_timestamp) as date, count(∗) as contracts_creation from ‘*bigquery − public − data.ethereum_blockchain.contracts*‘ as contracts where block_timestamp < *′*2019−05−21*′* group by date), b as (select date, sum(contracts_creation) over (order by date) as ccc , lead(date, 1) over (order by date) as next_date from a order by date), calendar as (select date from unnest(generate_date_array(*′*2015−07−30*′*, current_date())) as date), c as (select calendar.date, ccc from b join calendar on b.date <= calendar.date and calendar.date < b.next_date order by calendar.date) select date, ccc as cumulative_contract_creation from c order by date

- Number of contracts created by day (all contracts)
  select count(distinct(address)) as number_of_contracts, date(block_timestamp) as date

from ‘*bigquery−public−data.ethereum_blockchain.contracts*‘ where block_timestamp $< '2019-05-21'$ group by date order by date

- Total number of transactions in total as of today (allocation)
  with a as (select count(transactions.hash) as contract_calls
  from ‘*bigquery − public − data.ethereum_blockchain.transactions*‘ as transactions where input $!= '0x'$ and to_address is not null and block_timestamp $< '2019-05-21'$), b as (select count(transactions.hash) as contract_creation
  from ‘*bigquery − public − data.ethereum_blockchain.transactions*‘ as transactions where to_address is null and block_timestamp $< '2019-05-21'$), c as (select count(transactions.hash) as ether_transactions
  from ‘*bigquery − public − data.ethereum_blockchain.transactions*‘ as transactions where input $= '0x'$ and value is not NULL and block_timestamp $< '2019-05-21'$) select contract_calls, contract_creation, ether_transactions from a,b,c

- Number of transactions per day (comparison of contract calls and ether transfers)
  with a as (select date(block_timestamp) as date1, count(∗) as contract_calls
  from ‘*bigquery − public − data.ethereum_blockchain.transactions*‘ where input $!= '0x'$ and to_address is not null and block_timestamp $< '2019-05-21'$ group by date1), b as (select date(block_timestamp) as date2, count(∗) as contract_creation
  from ‘*bigquery−public−data.ethereum_blockchain.transactions*‘ where to_address is null and block_timestamp $< '2019-05-21'$ group by date2), c as (select date(block_timestamp) as date3, count(∗) as ether_transfers
  from ‘*bigquery − public − data.ethereum_blockchain.transactions*‘ where input $= '0x'$ and value is not NULL and block_timestamp $< '2019-05-21'$ group by date3) select contract_calls, ether_transfers, date1 from a join c on date1 = date3 order by date1

- Growth of contracts created directly by users
  with a as (select date(block_timestamp) as date, count(∗) as contracts_creation
  from ‘*bigquery−public−data.ethereum_blockchain.traces*‘ as traces where block_timestamp $< '2019-05-21'$ and trace_type $= 'create'$ and trace_address is null group by date), b as (select date, sum(contracts_creation) over (order by date) as ccc , lead(date, 1) over (order by date) as next_date from a order by date), calendar as (select date from unnest(generate_date_array($'2015-07-30'$, current_date())) as date), c as (select calendar.date, ccc from b join calendar on b.date <= calendar.date and calendar.date < b.next_date order by calendar.date) select date, ccc as cumulative_contract_creation from c order by date

- Growth of contracts created by other contracts
  with a as (select date(block_timestamp) as date, count(∗) as contracts_creation
  from ‘*bigquery−public−data.ethereum_blockchain.traces*‘ as traces where block_timestamp $< '2019-05-21'$ and trace_type $= 'create'$ and trace_address is not null group by date), b as (select date, sum(contracts_creation) over (order by date) as ccc ,

lead(date, 1) over (order by date) as next_date from a order by date), calendar as (select date from unnest(generate_date_array('2015 − 07 − 30', current_date())) as date), c as (select calendar.date, ccc from b join calendar on b.date <= calendar.date and calendar.date < b.next_date order by calendar.date) select date, ccc as cumulative_contract_creation from c order by date

- Combination of 15 and 16
  with a as (select date(block_timestamp) as date, count(∗) as contracts_creation
  from '*bigquery−public−data.ethereum_blockchain.traces*' as traces where block_timestamp
  < '2019 − 05 − 21' and trace_type = '*create*' and trace_address is null group by date),
  b as (select date, sum(contracts_creation) over (order by date) as ccc , lead(date, 1)
  over (order by date) as next_date from a order by date), calendar as (select date from
  unnest(generate_date_array('2015 − 07 − 30', current_date())) as date), c as (select
  calendar.date, ccc from b join calendar on b.date <= calendar.date and calendar.date
  < b.next_date order by calendar.date), d as (select date(block_timestamp) as date1,
  count(∗) as contracts_creation1
  from '*bigquery−public−data.ethereum_blockchain.traces*' as traces where block_timestamp
  < '2019 − 05 − 21' and trace_type = '*create*' and trace_address is not null group by
  date1), e as (select date1, sum(contracts_creation1) over (order by date1) as ccc1 ,
  lead(date1, 1) over (order by date1) as next_date1 from d order by date1), calendar1
  as (select date1 from unnest(generate_date_array('2015 − 07 − 30', current_date()))
  as date1), f as (select calendar1.date1, ccc1 from e join calendar1 on e.date1 <=
  calendar1.date1 and calendar1.date1 < e.next_date1 order by calendar1.date1) se-
  lect date1, date, f.ccc1 as cumulative_contract_creation_by_contracts, c.ccc as cumu-
  lative_contract_creation_by_users from c join f on f.date1 = c.date order by f.date1

- Contract creation transactions per day (directly from users)
  select date(block_timestamp) as date, count(∗) as contract_creation
  from '*bigquery − public − data.ethereum_blockchain.transactions*' where to_address
  is null and block_timestamp < '2019 − 05 − 21' group by date order by date

- Daily ether volume for ether transfers and contract calls
  with a as (select date(block_timestamp) as date, sum(value)/power(10,18) as con-
  tract_calls
  from '*bigquery − public − data.ethereum_blockchain.transactions*' where input ! =
  '0x' and to_address is not null and block_timestamp < '2019 − 05 − 21' group by
  date), b as (select date(block_timestamp) as date2, sum(value)/power(10,18) as con-
  tract_creation
  from '*bigquery−public−data.ethereum_blockchain.transactions*' where to_address is
  null and block_timestamp < '2019−05−21' group by date2), c as (select date(block_timestamp)
  as date3, sum(value)/power(10,18) as ether_transfers
  from '*bigquery − public − data.ethereum_blockchain.transactions*' where input =
  '0x' and value is not NULL and block_timestamp < '2019 − 05 − 21' group by date3)

select contract_calls, ether_transfers, contract_creation, date from a join c on date = date3 join b on date = date2 order by date

- Top 10 addresses measured by balance (value of ether)
  with value_table as ( select to_address as address, value as value
  from ‘*bigquery − public − data.ethereum_blockchain.traces*‘ where to_address is not null and block_timestamp < ′2019 − 05 − 21′ and status = 1 and (call_type not in (′*delegatecall*′, ′*callcode*′, ′*staticcall*′) or call_type is null) union all select from_address as address, -value as value
  from ‘*bigquery − public − data.ethereum_blockchain.traces*‘ where from_address is not null and block_timestamp < ′2019 − 05 − 21′ and status = 1 and (call_type not in (′*delegatecall*′, ′*callcode*′, ′*staticcall*′) or call_type is null) union all select miner as address, sum(cast(receipt_gas_used as numeric) ∗ cast(gas_price as numeric)) as value
  from ‘*bigquery − public − data.ethereum_blockchain.transactions*‘ as transactions
  join ‘*bigquery−public−data.ethereum_blockchain.blocks*‘ as blocks on blocks.number = transactions.block_number where block_timestamp < ′2019 − 05 − 21′ group by blocks.miner union all select from_address as address, -(cast(receipt_gas_used as numeric) ∗ cast(gas_price as numeric)) as value
  from ‘*bigquery−public−data.ethereum_blockchain.transactions*‘ where block_timestamp < ′2019 − 05 − 21′ ) select address, floor(sum(value)/power(10,18)) as balance from value_table group by address order by balance desc limit 10

- Top 10 contract addresses
  with value_table as ( select to_address as address, value as value
  from ‘*bigquery − public − data.ethereum_blockchain.traces*‘ where to_address is not null and block_timestamp < ′2019 − 05 − 21′ and status = 1 and (call_type not in (′*delegatecall*′, ′*callcode*′, ′*staticcall*′) or call_type is null) union all select from_address as address, -value as value
  from ‘*bigquery − public − data.ethereum_blockchain.traces*‘ where from_address is not null and block_timestamp < ′2019 − 05 − 21′ and status = 1 and (call_type not in (′*delegatecall*′, ′*callcode*′, ′*staticcall*′) or call_type is null) union all select miner as address, sum(cast(receipt_gas_used as numeric) ∗ cast(gas_price as numeric)) as value
  from ‘*bigquery − public − data.ethereum_blockchain.transactions*‘ as transactions
  join ‘*bigquery−public−data.ethereum_blockchain.blocks*‘ as blocks on blocks.number = transactions.block_number where block_timestamp < ′2019 − 05 − 21′ group by blocks.miner union all select from_address as address, -(cast(receipt_gas_used as numeric) ∗ cast(gas_price as numeric)) as value
  from ‘*bigquery−public−data.ethereum_blockchain.transactions*‘ where block_timestamp < ′2019−05−21′ ) select contracts.address, floor(sum(double_entry_book.value)/power(10,18)) as balance from value_table
  join ‘*bigquery − public − data.ethereum_blockchain.contracts*‘ as contracts

on value_table.address = contracts.address group by address order by balance desc limit 10

- Ether Balance distribution
  with value_table as ( select to_address as address, value as value
  from ‘*bigquery − public − data.ethereum_blockchain.traces*‘ where to_address is not null and block_timestamp $< '2019 - 05 - 21'$ and status = 1 and (call_type not in $('delegatecall', 'callcode', 'staticcall')$ or call_type is null) union all select from_address as address, -value as value
  from ‘*bigquery − public − data.ethereum_blockchain.traces*‘ where from_address is not null and block_timestamp $< '2019 - 05 - 21'$ and status = 1 and (call_type not in $('delegatecall', 'callcode', 'staticcall')$ or call_type is null) union all select miner as address, sum(cast(receipt_gas_used as numeric) ∗ cast(gas_price as numeric)) as value
  from ‘*bigquery − public − data.ethereum_blockchain.transactions*‘ as transactions
  join ‘*bigquery−public−data.ethereum_blockchain.blocks*‘ as blocks on blocks.number = transactions.block_number where block_timestamp $< '2019 - 05 - 21'$ group by blocks.miner union all select from_address as address, -(cast(receipt_gas_used as numeric) ∗ cast(gas_price as numeric)) as value
  from ‘*bigquery−public−data.ethereum_blockchain.transactions*‘ where block_timestamp $< '2019 - 05 - 21'$ ), a as (select sum(value)/power(10,18) as balance, address from value_table group by address) select balance, row_number() over (order by balance desc) as number_of_addresses from a order by balance desc

- Logarithmic ether balance distribution
  with value_table as ( select to_address as address, value as value
  from ‘*bigquery − public − data.ethereum_blockchain.traces*‘ where to_address is not null and block_timestamp $< '2019 - 05 - 21'$ and status = 1 and (call_type not in $('delegatecall', 'callcode', 'staticcall')$ or call_type is null) union all select from_address as address, -value as value
  from ‘*bigquery − public − data.ethereum_blockchain.traces*‘ where from_address is not null and block_timestamp $< '2019 - 05 - 21'$ and status = 1 and (call_type not in $('delegatecall', 'callcode', 'staticcall')$ or call_type is null) union all select miner as address, sum(cast(receipt_gas_used as numeric) ∗ cast(gas_price as numeric)) as value
  from ‘*bigquery − public − data.ethereum_blockchain.transactions*‘ as transactions
  join ‘*bigquery−public−data.ethereum_blockchain.blocks*‘ as blocks on blocks.number = transactions.block_number where block_timestamp $< '2019 - 05 - 21'$ group by blocks.miner union all select from_address as address, -(cast(receipt_gas_used as numeric) ∗ cast(gas_price as numeric)) as value
  from ‘*bigquery−public−data.ethereum_blockchain.transactions*‘ where block_timestamp $< '2019 - 05 - 21'$ ), a as (select sum(value)/power(10,18) as balance, address from value_table group by address), b as (select balance, row_number() over (order by

balance desc) as address_rank from a where balance > 0), c as (select balance, floor(log(address_rank) * 100) as log_address_rank, address_rank from b) select avg(balance) as balance_avg, log_address_rank, count(address_rank) as number_of_addresses from c group by log_address_rank order by balance_avg desc

- Logarithmic ether transactions distribution
with value_table as ( select to_address as address, count(distinct(transactions.hash)) as tx
from '*bigquery − public − data.ethereum_blockchain.transactions*' as transactions where to_address is not null and block_timestamp < '2019 − 05 − 21' group by address union all select from_address as address, count(distinct(transactions.hash)) as tx
from '*bigquery − public − data.ethereum_blockchain.transactions*' as transactions where from_address is not null and block_timestamp < '2019 − 05 − 21' group by address ), a as (select sum(tx) as transactions, address from value_table group by address), b as (select transactions, row_number() over (order by transactions desc) as address_rank from a), c as (select transactions, floor(log(address_rank) * 100) as log_address_rank, address_rank from b) select avg(transactions) as transactions_avg, log_address_rank, count(address_rank) as number_of_addresses from c group by log_address_rank order by transactions_avg desc

- Logarithmic ether transactions distribution in recipients addresses (similar for sender addresses)
with a as ( select to_address as address, count(distinct(transactions.hash)) as tx
from '*bigquery − public − data.ethereum_blockchain.transactions*' as transactions where to_address is not null and block_timestamp < '2019 − 05 − 21' group by address), b as (select tx, row_number() over (order by tx desc) as address_rank from a), c as (select tx, floor(log(address_rank) * 100) as log_address_rank, address_rank from b) select avg(tx) as transactions_avg, log_address_rank, count(address_rank) as number_of_addresses from c group by log_address_rank order by transactions_avg desc

- Top 1000 ether holders by transaction activity and balance
with value_table as ( select to_address as address, value as value
from '*bigquery − public − data.ethereum_blockchain.traces*' where to_address is not null and block_timestamp < '2019 − 05 − 21' and status = 1 and (call_type not in ('*delegatecall*', '*callcode*', '*staticcall*') or call_type is null) union all select from_address as address, -value as value
from '*bigquery − public − data.ethereum_blockchain.traces*' where from_address is not null and block_timestamp < '2019 − 05 − 21' and status = 1 and (call_type not in ('*delegatecall*', '*callcode*', '*staticcall*') or call_type is null) union all select miner as address, sum(cast(receipt_gas_used as numeric) * cast(gas_price as numeric)) as value
from '*bigquery − public − data.ethereum_blockchain.transactions*' as transactions join '*bigquery−public−data.ethereum_blockchain.blocks*' as blocks on blocks.number

= transactions.block_number where block_timestamp < '2019 − 05 − 21' group by blocks.miner union all select from_address as address, -(cast(receipt_gas_used as numeric) ∗ cast(gas_price as numeric)) as value
from '*bigquery−public−data.ethereum_blockchain.transactions*' where block_timestamp < '2019 − 05 − 21' ), a as (select sum(value)/power(10,18) as balance, address from value_table group by address order by balance desc),
b as (select to_address, count(transactions.hash) as tx_recipient from '*bigquery − public−data.ethereum_blockchain.transactions*' as transactions where block_timestamp < '2019−05−21' group by to_address), c as (select from_address, count(transactions.hash) as tx_sender
from '*bigquery − public − data.ethereum_blockchain.transactions*' as transactions where block_timestamp < '2019 − 05 − 21' group by from_address) select ∗ from a left join b on (a.address = b.to_address) left join c on (a.address = c.from_address) order by balance desc limit 1000

- Top 1000 addresses measured by number of received transactions
with value_table as ( select to_address as address, value as value
from '*bigquery − public − data.ethereum_blockchain.traces*' where to_address is not null and block_timestamp < '2019 − 05 − 21' and status = 1 and (call_type not in ('*delegatecall*', '*callcode*', '*staticcall*') or call_type is null) union all select from_address as address, -value as value
from '*bigquery − public − data.ethereum_blockchain.traces*' where from_address is not null and block_timestamp < '2019 − 05 − 21' and status = 1 and (call_type not in ('*delegatecall*', '*callcode*', '*staticcall*') or call_type is null) union all select miner as address, sum(cast(receipt_gas_used as numeric) ∗ cast(gas_price as numeric)) as value
from '*bigquery − public − data.ethereum_blockchain.transactions*' as transactions join '*bigquery−public−data.ethereum_blockchain.blocks*' as blocks on blocks.number = transactions.block_number where block_timestamp < '2019 − 05 − 21' group by blocks.miner union all select from_address as address, -(cast(receipt_gas_used as numeric) ∗ cast(gas_price as numeric)) as value
from '*bigquery−public−data.ethereum_blockchain.transactions*' where block_timestamp < '2019 − 05 − 21' ), a as (select sum(value)/power(10,18) as balance, address from value_table group by address order by balance desc),
b as (select to_address, count(transactions.hash) as tx_recipient from '*bigquery − public−data.ethereum_blockchain.transactions*' as transactions where block_timestamp < '2019−05−21' group by to_address), c as (select from_address, count(transactions.hash) as tx_sender
from '*bigquery − public − data.ethereum_blockchain.transactions*' as transactions where block_timestamp < '2019 − 05 − 21' group by from_address) select to_address, tx_recipient, balance from b left join a on (a.address = b.to_address) order by tx_recipient desc limit 1000

- Top 1000 addresses measured by number of sent transactions
  with value_table as ( select to_address as address, value as value
  from '*bigquery − public − data.ethereum_blockchain.traces*' where to_address is not null and block_timestamp < '2019 − 05 − 21' and status = 1 and (call_type not in ('*delegatecall*', '*callcode*', '*staticcall*') or call_type is null) union all select from_address as address, -value as value
  from '*bigquery − public − data.ethereum_blockchain.traces*' where from_address is not null and block_timestamp < '2019 − 05 − 21' and status = 1 and (call_type not in ('*delegatecall*', '*callcode*', '*staticcall*') or call_type is null) union all select miner as address, sum(cast(receipt_gas_used as numeric) ∗ cast(gas_price as numeric)) as value
  from '*bigquery − public − data.ethereum_blockchain.transactions*' as transactions join '*bigquery−public−data.ethereum_blockchain.blocks*' as blocks on blocks.number = transactions.block_number where block_timestamp < '2019 − 05 − 21' group by blocks.miner union all select from_address as address, -(cast(receipt_gas_used as numeric) ∗ cast(gas_price as numeric)) as value
  from '*bigquery−public−data.ethereum_blockchain.transactions*' where block_timestamp < '2019 − 05 − 21' ), a as (select sum(value)/power(10,18) as balance, address from value_table group by address order by balance desc),
  b as (select to_address, count(transactions.hash) as tx_recipient from '*bigquery − public−data.ethereum_blockchain.transactions*' as transactions where block_timestamp < '2019−05−21' group by to_address), c as (select from_address, count(transactions.hash) as tx_sender
  from '*bigquery − public − data.ethereum_blockchain.transactions*' as transactions where block_timestamp < '2019−05−21' group by from_address) select from_address, tx_sender, balance from c left join a on (a.address = c.from_address) order by tx_sender desc limit 1000

- Number of unique bytecodes of user-created contracts
  select distinct(output) as unique_bytecodes
  from '*bigquery−public−data.ethereum_blockchain.traces*' as traces where block_timestamp < '2019 − 05 − 21' and trace_type = '*create*' and trace_address is null

- Number of unique bytecodes of contract-created contracts
  select distinct(output) as unique_bytecodes
  from '*bigquery−public−data.ethereum_blockchain.traces*' as traces where block_timestamp < '2019 − 05 − 21' and trace_type = '*create*' and trace_address is not null

**Exact addresses that are used in table 4.3**

| Shortened Address | Address |
|---|---|
| 0xc0...cc2 | 0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2 |
| 0x4e...a67 | 0x4e9ce36e442e55ecd9025b9a6e0d88485d628a67 |
| 0x74...44e | 0x742d35cc6634c0532925a3b844bc454e4438f44e |
| 0x53...a3d | 0x53d284357ec70ce289d6d64134dfac8e511c8a3d |
| 0x66...054 | 0x66f820a414680b5bcda5eeca5dea238543f42054 |
| 0xab...83e | 0xab7c74abc0c4d48d1bdad5dcb26153fc8780f83e |
| 0x61...eea | 0x61edcdf5bb737adffe5043706e7c5bb1f1a56eea |
| 0xdc...0d3 | 0xdc76cd25977e0a5ae17155770273ad58648900d3 |
| 0xfc...8e4 | 0xfca70e67b3f93f679992cd36323eeb5a5370c8e4 |
| 0xe8...919 | 0xe853c56864a2ebe4576a807d26fdc4a0ada51919 |
| 0x22...0f6 | 0x229b5c097f9b35009ca1321ad2034d4b3d5070f6 |
| 0xde...bae | 0xde0b295669a9fd93d5f28d9ec85e40f4cb697bae |
| 0x13...9c5 | 0x137ad9c4777e1d36e4b605e745e8f37b2b62e9c5 |
| 0x1b...7c2 | 0x1b3cb81e51011b549d78bf720b0d924ac763a7c2 |
| 0x7e...164 | 0x7ef35bb398e0416b81b019fea395219b65c52164 |
| 0x26...dc0 | 0x267be1c1d684f78cb4f6a176c4911b741e4ffdc0 |
| 0x51...0eb | 0x51f9c432a4e59ac86282d6adab4c2eb8919160eb |
| 0x84...0ed | 0x847ed5f2e5dde85ea2b685edab5f1f348fb140ed |
| 0x6c...a7b | 0x6cc5f688a315f3dc28a7781717a9a798a59fda7b |
| 0x74...e82 | 0x74660414dfae86b196452497a4332bd0e6611e82 |

**Exact addresses that are used in table 4.5**

| Shortened Address | Address |
|---|---|
| 0xea...ec8 | 0xea674fdde714fd979de3edf0f56aa9716b898ec8 |
| 0x52...3b5 | 0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5 |
| 0xfb...b98 | 0xfbb1b73c4f0bda4f67dca266ce6ef42f520fbb98 |
| 0x82...830 | 0x829bd824b016326a401d083b33d092293333a830 |
| 0x5a...c4c | 0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c |
| 0x2a...226 | 0x2a65aca4d5fc5b5c859090a6c34d164135398226 |
| 0xa7...49e | 0xa7a7899d944fe658c4b0a1803bab2f490bd3849e |
| 0x3f...0b3 | 0x3f5ce5fbfe3e9af3971dd833d26ba9b5c936f0be |
| 0xce...a32 | 0xceceaa8edc0830c7cec497e33bb3a3c28dd55a32 |
| 0xb2...347 | 0xb2930b35844a230f00e51431acae96fe543a0347 |
| 0x6c...21f | 0x6cace0528324a8afc2b157ceba3cdd2a27c4e21f |
| 0x2b...258 | 0x2b5634c42055806a59e9107ed44d43c426e58258 |
| 0xd5...2ff | 0xd551234ae421e3bcba99a0da6d736074f22192ff |
| 0x06...bbf | 0x0681d8db095565fe8a346fa0277bffde9c0edbbf |
| 0x56...ced | 0x564286362092d8e7936f0549571a803b203aaced |
| 0x32...d88 | 0x32be343b94f860124dc4fee278fdcbd38c102d88 |
| 0x0d...2fe | 0x0d0707963952f2fba59dd06f2b425ace40b492fe |
| 0xd3...375 | 0xd34da389374caad1a048fbdc4569aae33fd5a375 |
| 0x26...dc0 | 0x267be1c1d684f78cb4f6a176c4911b741e4ffdc0 |
| 0x61...bd9 | 0x61c808d82a3ac53231750dadc13c777b59310bd9 |

**Exact addresses that are used in table 4.6**

| Shortened Address | Address |
|---|---|
| 0x8d...819 | 0x8d12a197cb00d4747a1fe03395095ce2a5cc6819 |
| 0x2a...208 | 0x2a0c0dbecc7e4d658f48e01e3fa353f44050c208 |
| 0x3f...0be | 0x3f5ce5fbfe3e9af3971dd833d26ba9b5c936f0be |
| 0x06...66d | 0x06012c8cf97bead5deae237070f9587f8e7a266d |
| 0x86...db0 | 0x86fa049857e0209aa7d9e616f7eb3b3b78ecfdb0 |
| 0xf2...2e2 | 0xf230b790e05390fc8295f4d3f60332c93bed42e2 |
| 0x70...413 | 0x70faa28a6b8d6829a4b1e649d26ec9a2a39ba413 |
| 0x20...8ef | 0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef |
| 0xd1...405 | 0xd1ceeeeee83f8bcf3bedad437202b6154e9f5405 |
| 0xe9...ff3 | 0xe94b04a0fed112f3664e45adb2b8915693dd5ff3 |
| 0xa3...770 | 0xa3c1e324ca1ce40db73ed6026c4a177f099b5770 |
| 0xd2...c07 | 0xd26114cd6ee289accf82350c8d8487fedb8a0c07 |
| 0x60...76b | 0x60d0cc2ae15859f69bf74dadb8ae3bd58434976b |
| 0xf5...4f3 | 0xf5bec430576ff1b82e44ddb5a1c93f6f9d0884f3 |
| 0x03...b0e | 0x03df4c372a29376d2c8df33a1b5f001cd8d68b0e |
| 0xfa...0b3 | 0xfa52274dd61e1643d2205169732f29114bc240b3 |
| 0x60...8ef | 0x6090a6e47849629b7245dfa1ca21d94cd15878ef |
| 0x0e...50f | 0x0e50e6d6bb434938d8fe670a2d7a14cd128eb50f |
| 0x17...ab9 | 0x174bfa6600bf90c885c7c01c7031389ed1461ab9 |