***Abstract*—**
***Index Terms*—**

Analyzing the Ethereum Blockchain

Christin Sauerbier
Humboldt University Berlin
Berlin, Germany
christin.sauerbier@outlook.com

Roman Proskalovich
Humboldt University Berlin
Berlin, Germany
romanarion@gmail.com

Thomas Siskos
Humboldt University Berlin
Berlin, Germany
thomas.siskos@hu-berlin.de

I. Introduction

With the Introduction of Bitcoin in 2008, a signifikant new currency was brought into life. It is based on a peer to peer network which gives the opportunity to cancel out intermediaries within money transfers, through a decentralized verification process. With that knowledge the idea was born to use such a technology to run decentralized applications on such a platform. Ethereum can provide that and is now the second largest cryptocurrency after Bitcoin in terms of market capitalization. []

This paper is supposed to do an explorative study on the Ethereum blockchain. The idea is to replicate what Lischke and Fabian [Lischke and Fabian, 2016] did and transfer it to another crypto technology the Ethereum Blockchain. They used the address spaces and transaction history of the Bitcoin and added off-set network data to get more insights about the bitcoin economy. The period analyzed, included the first four years of the bitcoin existence. They were able to combine off-set network data and transaction data to get good statements about the business categories the currency is used for. With the Ethereum we try to figure out if we are able to collect the transaction data and IP addresses to get more insights about the crypto technology, more detailed it we want to figure out how much information on is able to get out of the transaction and off set data to get deeper insights about what is going on in the network and does these information contain security problems for the network. The period is shorter than it was done in the replicated paper, but we do used the same resources.

This paper is structured as follows. Section II and III are dedicated to the basics of Ethereum and how it got evolved from the bitcoin currency, as well as the technology behind it, respectively. As we replicate a paper about the bitcoin blockchain [Fabian, 2016] we also compare the Ethereum blockchain with the bitcoin blockchain. We follow this by a brief literature review in section IV.

After that the reader got a proper foundation that we can dive into the underlying methodology in section V. The focus here lies mainly on how we obtained our dataset in subsection V-A and an overview of the methods used during graph analysis in subsection V-B. Moving onward we present our findings in section VI, where we describe the outcome of our methods for data aquiration in subsection VI-A, structure them using traditional statistical methods in subsection VI-B and graph analysis in subsection VI-C. Finally, section VII concludes the article and discusses the outlook we received during our analysis. We also want to use this part to discuss what are the main differences to the former paper about the bitcoin blockchain and give an idea about future work.

II. Ethereum Theory

Before diving into Ethereum one has to give a short introduction about the main invention of cryptocurrencies, Bitcoin and the blockchain technology behind it. These ideas drove Vitalik Buterin when he tried to design another cryptocurrency to take the bitcoin technology to the next level by giving developers to opportunity to write their own programs on the blockchain technology. This created the opportunity to use the blockchain technology and combine it with several business cases, not just as a currency.

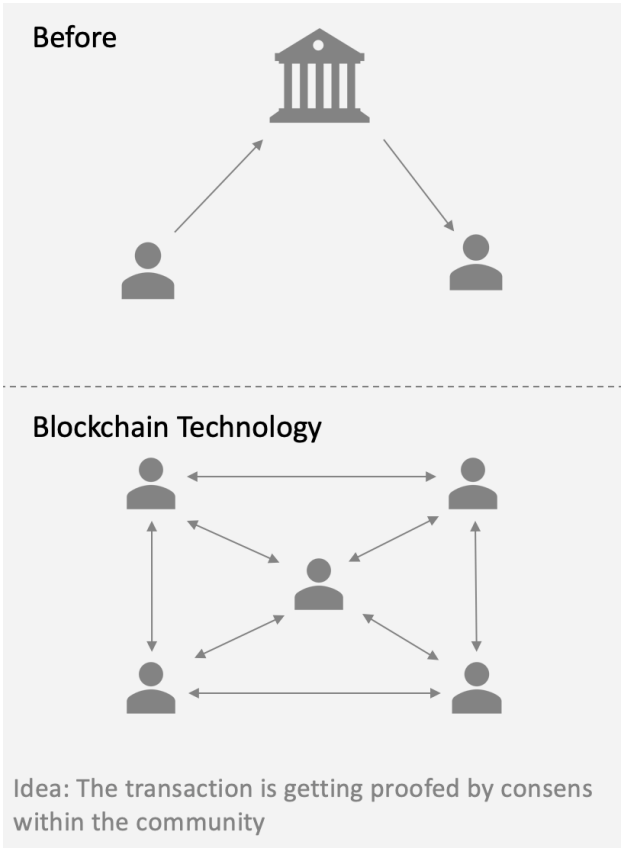
The Bitcoin is a decentralized cryptocurrency which was invented by Satoshi Nakamoto, it combines three technologies with each other, Cryptography, Proof of Work and decentralized Networks. The idea was to create a currency which is independent from other institutions, who control the network as an intermediary. When using cryptocurrency, the network is decentralized, and an intermediary becomes obsolete. The network takes over the function of an intermediary, as everyone keeps a transaction history, and this gets verified against each other, every time someone makes a new entry. This results in a technology that is hard to manipulate, meaning that it becomes increasingly hard for someone to take over, control or shut down the entire network [Grishchenko et al., 2018].

They could realize such a network with three already existing technologies which got combined the first time by Satoshi Nakamoto, for the Bitcoin. It describes the whole process how the Bitcoin blockchain gets assembled, validated and agreed for a transaction. The blockchain can be assumed as a database that is stored on several computers, in the same version [Grishchenko et al., 2018].

The blockchain component can be described as a series of blocks that are combined ordered or chained with each other. Everyone stores a variety of information with a hash function. Were the information in every hash is related to the former one, plus the verification that the transaction is proofed. So far, most crypto technologys work with the so-called proof-of-work [Ray, 2018].

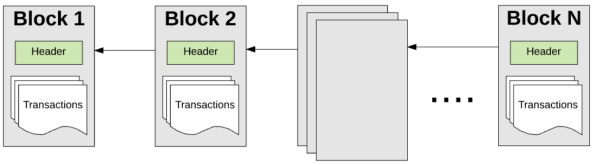
The main idea is to find a consent in the network which transactions or blocks are getting approved trough the network. The work part in proof of work relates to the puzzle the

Fig. 1: Blockchaintechnology is the basis for Ethereum



community has to solve to verify a transaction and if enough transactions get combined, they can get inserted into a block.

Fig. 2: How Blocks get chained



This means that transactions can only be verified, if they are gathered and inserted into a newly created block. The person in the network who is creating the new block is called miner and gets compensated with a value for mining. The exact compensation depends on the crypto technology. In Bitcoin if they verify for the Bitcoin network and in gas if they are mining for the Ethereum Blockchain. Every transaction gets collected through it's hash and approved in a block which line up to a chain, so that it gets computationally hard to manipulate a transaction. Here one can lead over to the next important component of the bitcoin technology, the peer to peer networks.

Transferred it means, that there is a network in which each peer is connected to each other without any other intermediary in between. There is a direct connection between the participants within the network. It also is decentralized, such that nobody can manipulate anything.

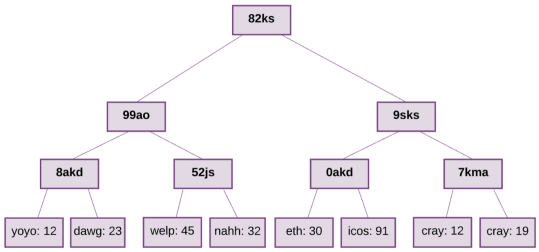
When the creator of Ethereum thought about this idea and technology, the main question was, what else can be decentralized to cancel trusted intermediaries out and go beyond currencies. This let Vitalik Buterin in his white paper in late 2013, to describe a platform which can run decentralized applications and therefore run economic systems in pure software. The Vision of Ethereum is to create an unstoppable censorship-resistant self-sustaining decentralized world computer. [Ray, 2018]. The Ethereum can be explained best, if the dimensions get differentiated into the currency and the platform. The platform includes the Ethereum Virtual Machine, the smart contracts and the decentralized applications.

To add on to the former Bitcoin explanations, the Ethereum is also providing a virtual machine to realize smart contract code. Before diving into that topic it should be noted that the Ethereum platform is able to access and alter data to compute anything a turing complete machine could, it is also able to use conditional branches. The last requirement to fulfill the turing completeness is to have an arbitrary amount of memory. As not all requirements are met the Ethereum virtual machine (EVM) is only quasi-turing complete, because every transaction or computation is requiring a defined amount of gas, that is used to pay the system for mining the blocks. The programming language to write applications for the EVM is solidity, which gets compiled to the EVM bytecode. What is the EVM programming language? The advantage to work on the Ethereum virtual machine allows to make transitions from one state to the other. For example, transactions get executed and gathered into a new block, the new state gets stored and this new block describes the next state. Here Merkle Patricia trees are used to optimize the verification process of transactions on the EVM [Bisade, 2018].

The mapping between account addresses and the equivalent state of the account has to get stored within the Ethereum platform. That way the data doesnt need to be sored in a giant block header. The Merkle Tree structure is thus used to make the hole system scalable. In general, a Merkle is following the idea to have a big amount of data which are hashed together. They get split into smaller chunk buckets, this gets repeated until there is just a root hash left [Butin, 2015].

As stated before, this whole process is done to make it possible to perform Merkle proofs. They are much quicker and therefore better scalable. The proof consists of all the hashes going from the last one up to the root hash, if this get verified by other persons, at least for the verified branch. The idea is, that within the database the data is stored in a Merkle tree and the main root is publicly known and proofed. If so, a person who wants to get the knowledge about data at a specific position this can be done within less time. Because just small amounts of data have to get searched [Buterin, 2015]. This capability to store information in Merkle trees is very useful to improve the scalability and it is called light nodes. So, in the blockchain system exists to types of nodes, full and light ones. Whereas the complete one does synchronize the hole

Fig. 3: Merkle Tree Structure

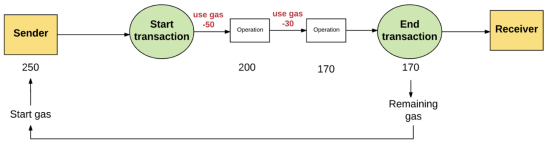


chain and is also executing everyone. Even though a node does keep every transaction, it is not necessary to store all the data. One would just get the header of every transaction done since the first block was mined. This system works, because the data is stored in merle trees and can get verified top down. In conclusion, the advantages of using such a tree is that the hashes in one tree always dependent on the other hash buckets in the rest of the tree. Therefore, if someone wants to manipulate something, it is hard to accomplish. As the root hash always keeps information about actual the state, the receipts and the completed transactions, nodes can validate little parts of the chain [Kasireddz 2017].

After giving insights into the processing and storing process in the Ethereum World, the applications on the platform are another important topic. The Smart contract can be used to run code on the Ethereum Virtual machine in a decentralized way. It is comparable to real world contracts. A collection of conditions and actions, if a condition occurs, the associated actions are getting executed. As a result many real-world applications can be applied to smart contracts. The execution happens on the blockchain and is immutable ever after. One example often used, is a rent contract in the real work. In the real work one would pay rent on a defined date and get entrance to a place for the paid time. A contract represented on Ethereum would look the same, the contracts gets written in Solidity. Then the network executes the contract on the defined date and if the renter does pay as expected, means fulfill the condition he gets entrance to his place. If not, the contracts would simply not open the door anymore.

So, every component of a contract gets represented the enforcement, management and the payment component []. Back to the example, the person who is renting the property would not get access to his door anymore, if the rent was paid a day later. Here the first problem with smart contracts occurs, the rules are really strict, without any room for exceptional circumstances. Once the code is executed on the network it cannot be edited or corrected anymore. Everything gets executed as written in the code. If one does think about complex contracts this can be a problem. If something gets wrong or if there are paragraphs in it which are not covered by law, with a real contract the law in backing these contracts and one could take out paragraphs after a legal process. This is not the case in Ethereum, because to edit such a contract the

Fig. 4: Transactions



entire network would have to get convinced, that something hast to get changed. Which is almost impossible [].

Ethereum also provides a currency to run and incentivize the previously described system. In Ethereum the currency is called Ether. To execute and deploy contracts on in the platform and to fulfill the contracts a fee has to get payed. This will be done in Gas. The concept can be subdivided into gas price and gas. The gas-price represents the amount of Ether that one is willing to pay for every unit of gas. The subunit of Ether is Wei and is representing the smallest unit. The Gas serves as a fee that is required to execute something in the network. As shown in the figure, every operation spends the defined amount of gas. In case the gas is not enough to fulfill the transactions, the sender runs out of gas and the End transaction does not get reached.

If someone wants to execute something in the network, one has to decide about the gas limit and gas price. This has an influence on the amount of computations that one transaction can do. The miner can collect transactions from the transaction Pool, after they pick them their respective code gets executed on the Virtual Machine and the Merkle proof gets created. The last step is to confirm the transaction with the proof of work algorithm. The reward depends on the selected amount of gas for the transaction. That also means, that some transaction might not get mined in case the fee is chosen to low. Another part the gas is used for, is the storing fee, the amount depends on the size of the storage used []. On the one hand this was done, so that people will write optimized and efficient code and won‘t waste the Ethereum network computing power on unnecessary tasks []. On the other hand, this is a way to incentivize the mining process of transaction. This helps to verify the transaction and should be described a little more into detail. This process verifies the transaction and is carried out to secure against counterfeiting. It is also applying the proof-of-work and uses the Ethereum network, respectively

the nodes and creates new blocks with validated transactions.

The technology also has downsides, one is definitively the scalability. Every node in the system is storing the full data and executing every transaction. To store the state is a matter of security, also the high degree of decentralization. As more traffic gets on the system and more applications get written the data that have to be stored and verified for in the chain, get more and more. Specially on the Ethereum platform were there are more then just currency transaction other than the Bitcoin. To be comparable to common systems, the Ethereum blockchain has to be able to keep on to the security and decentralization but also work on the scalability. So far, the community is working on different alternatives, such as *sharding* and *plasma chains* [Gadaleta 2018].

Vitalik Buterin describes Sharding, combined with Plasma as the most promising solution to make the platform more scalable [Giese 2018]. Here Sharding describes a partition of the main net in smaller shards. So far, the blockchain can support 15 transactions per second, with the new solution this could work for 100 transactions afterwards. Combined with the plasma solution this could scale up to a million transactions per second. The plasma solution represents a second-layer solution could bundle transactions and handle therefore more at ones [Giese 2018]. The main issue always is, that if one wants to higher the scalability, the verifications has to be reduced or less difficult. In this triangle one always reducing something, what means the points which have to be manipulated in the network are getting reduced. Next problem is the high computational power that is needed to verify or mine the transactions, here a lot of energy is needed that the system can work. So far, there is no solution for that.

But with the change to the proof-of-stake this should change. Besides the scalability and the energy consumption Ethereum has to deal with the high volatility in the crypto market. One can imagine, that in case of Ethereum the volatility has an influence on the usage of the platform. As described before the currency, called Ether is meant to be the Gas (fee) to pay the system for executing applications. If the price went up and down and is opposed by speculation, the original purpose might not get used anymore [Chandersekhar 2018]. Chandersekhar looked into the data of Smart Contracts and found out, that 94 % of the smart contracts are called less then 10 times and just 5 % are getting called between 10 and 100 times.

III. Ethereum and the Blockchain

Bitcoin which was 2008 introduced, is the more mature platform, compared to Ethereum. We want to give a short overview about both technologies and compare them. As the original paper is analyzing and describing the Bitcoin world. On the one hand we want to give some insights in the most important technologies on the blockchain market and also might gather some information about possible problems with the outcome.

Lets start with the data, by now the Bitcoin blockchain has the size of 197 gigabytes in the beginning of January its growing constantly, since it was introduced [Portal, 2019b]. It takes on average 9,6 minutes to confirm a transaction at the end of February 2019 [Portal, 2019b]. The Bitcoin index value for the same period was 3.799,68 US Dollar, were the number of wallets grew over the last years constantly and reached 32 million wallet users at the end of 2018.

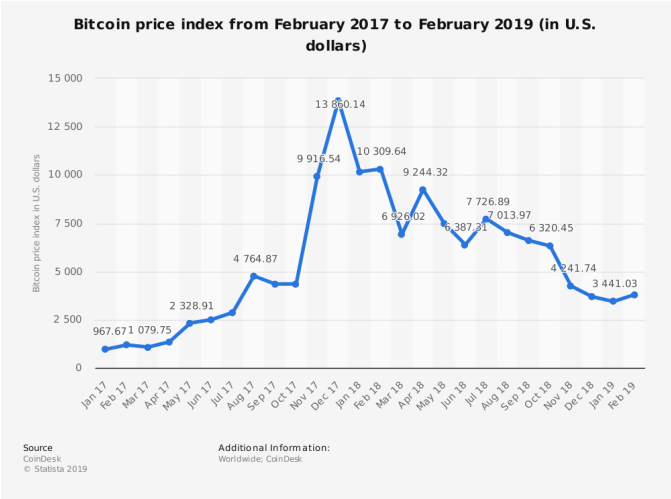


Fig. 5: Bitcoin Price Index- [**?**]

The market capitalization had a peak in Quarter four 2017 and it has since decreased, to a value of 66,18 million US Dollar in the end of 2018. Compared the Ethereum already grew to the size of 181.65 GB, this represents the full blockchain size of all blocks. The average mining time is less that the Bitcoin, with 13.5 s per block [BitInfoCharts, 2019]. The number of wallets exceeded 50 million in the end of 2018 [**?**].

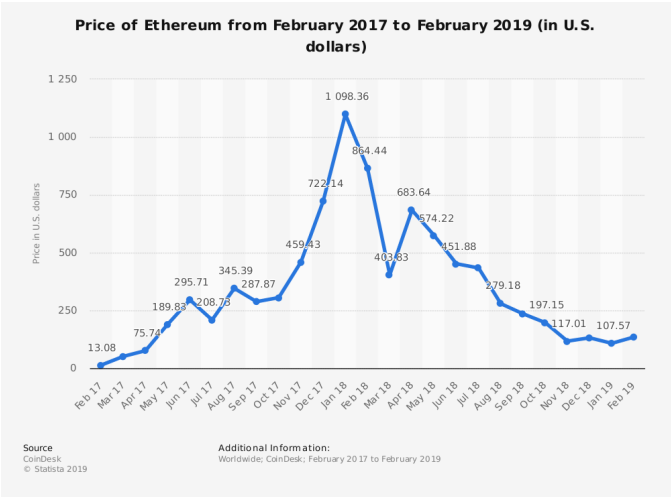


Fig. 6: Ethereum Price Development - [**?**]

Even if the wallets are increasing, the number of active addresses, referencing to BitInfoCharts, there are less active Ethereum addresses in 2018. The price of Ethereum is in the End of February 2019 amounted to 134,82 US Dollar [Portal,

2019a]. Comparable to the Bitcoin, the peak was in 2017 and has been decreasing since then. Similar to the peak in price to most active period was in the end of 2017, here more than 1.13 million addresses were registered [Partz,]. In the End of 2018 there were only 328.400 active addresses recorded.

The data can give some insights, but just in relation to the idea of both technologies they get more relevance. The most important difference is, that the idea behind the bitcoin was to create an independent currency. That is not easy to manipulate and does not need an intermediary to be trusted. The community should be the verification, that no one would manipulate the blockchain. The technology was described in the former chapter. In contrast the Ethereum was developed afterwards. The person who wrote the white paper, Vitalik Butin, wanted to create more than a currency. [Buterik, 2013] He got inspired by the technology behind the bitcoin and wanted to go beyond. That why he created a platform usable to write applications on, backed with the same technology. As described in the Ethereum Theory chapter, along the way he also was able to improve some downsides about the bitcoin, introducing the merkle Patricia tree. Another advantage is that he made the platform able to work with a turin-complete language. In the End all that was necessary, as the Ethereum Blockchain has to deal with way more complexity when it wants to serve as a platform for applications.

IV. Literature Review

We aspire to adopt the methods used in the study of blockchain described in [Lischke and Fabian, 2016]. The main difference of our research is that it targets ethereum instead of the bitcoin. For this reason we pay special attention to the papers focused on ethereum blockchain. Thus, [Payette et al., 2017] research Ethereum address space; [Chan and Olmsted, 2017] and [Chen et al., 2018] perform Ethereum graph analysis; [Li et al., 2017] develop a query layer for the Ethereum blockchain; [Anoaica and Levard, 2018] perform quantitative description of internal activity on the Ethereum blockchain; [Somin et al., 2018] study the dynamics of the social signals of Ethereum network, provide insights about the ecosystem and the forces acting within it and demonstrate that the network displays strong power-law properties.

We also go through the generic information about Ethereum: including its white [Buterin et al., 2014]) and yellow ([Wood, 2014] papers), comparative studies of crypto-currencies (see for instance [Maesa, 2018], [Rudlang, 2017], [Sapuric et al., 2017], [Anderson et al., 2016]). Given the role that smart-contracts play in Ethereum blockchain we also consider research on this topic. For instance: [Grishchenko et al., 2018] do semantic analysis of Ethereum smart-contracts; [Tikhomirov et al., 2018] come up with tools for their static analysis; [Bartoletti et al., 2017] research practical applications of smart contracts and their design patterns; [Bartoletti et al., 2017] suggest ways to identify Ponzi schemes.

To complete the picture we also consider research on network analysis of bitcoin blockchain that were suggested in [Lischke and Fabian, 2016] and can provide some additional

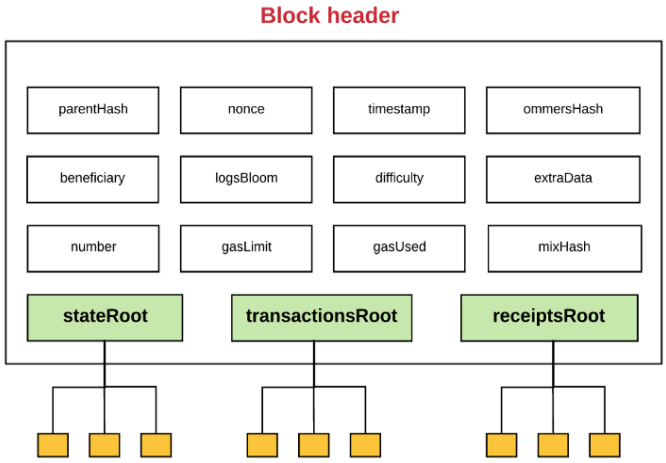


Fig. 7: Block headers [Preethi, 2017]

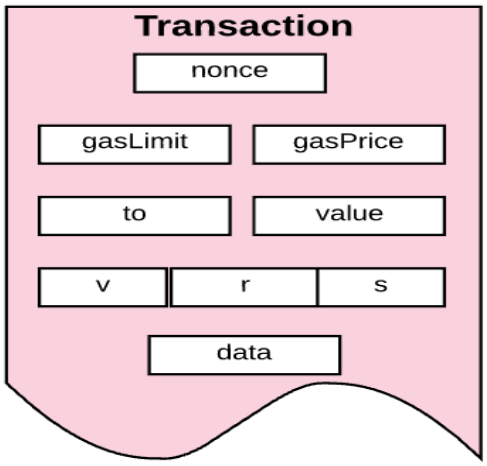


Fig. 8: Transaction contents [Preethi, 2017]

inspiration: [Reid and Harrigan, 2013], [Baumann et al., 2014], [Drainville, 2012], [Ober et al., 2013], [Meiklejohn et al., 2013], [Spagnuolo et al., 2014], [Androulaki et al., 2013], [Kaminsky, 2011], [Ortega, 2013].

V. Methodology

A. Data Preparation

In order to access the data in any form of blockchain technology one must first be an owner of a full node of the respective network. This means that the whole blockchain data with all its records is stored on the researcher's device. However for the most part, data in this form is not stored in a manner that is easy to query. For this reason we opted to become owners of a full node of the Ethereum blockchain, to subsequently scrape the blockchain for the desired information

and to store the results in a relational database, which in turn is easy to query.

The data-scraper is written in the `Python` language using the `web3.py` module [Rossum, 1995], [Piper and Caver, 2018]. Through `web3.py`'s interface it is possible to interact with the Ethereum blockchain. In particular the `web3.py` module enables us to extract information for every block and each transaction that is recorded in the blockchain.

Algorithm 1 Data Scraper: Overview

```
Check progress file if the ETL can be resumed from a block.
if yes then
    current block c ← content of progress file
else
    c ← 1
end if
s ← 2000000
while c ≤ s do
    Overwrite progress file with c
    content_Block ← getBlock(c)
    row_Block ← extractInfo(content_Block)
    write(row_Blocks, "Block")
    for each transaction in content_Block do
        row_Transaction ← extractInfo(transaction)
        write(row_Transaction, "Transactions")
    end for
end while
```

In particular, for each block inside a predetermined range we query its contents and obtain the data that is interesting for the further analysis. On the block level we extract the hash of each block as a unique identifier and the time each block was inserted into the blockchain. Note that this is not necessarily the time that the transaction was being commissioned. Finally we query the total gas that was used for each block as well as the gas limit for each block. Along with that information we retrieve a list of transactions that are contained within that block.

Making use of the list of transactions within a block we extract the wallet addresses of both sender and receiver, as well as the total value, of the transaction. Additionally we retrieve information on the gas price for the transaction. Figure 9 summarizes the relations within that database and Algorithm 1 gives an outline of the datascraper in pseudocode.

B. Graph Analysis

We can characterize a network by a number of metrics. The degree of a node in the network is the number of incoming or outgoing edges. We define a *triplet* i, v, j as an ordered set of three nodes, where v is the focal point and the undirected edges $< i, v >$ and $< v, j >$ form the neighboring edges. A triplet is considered closed, when there are exactly three connections for the three nodes. Three closed triplets form a *triangle* [Meghanathan, 2018]. Additionally, we can define multiple measures to investigate a node's role inside the network.

1) Degree Centrality: The most common centrality measure focuses on counting the number of nodes in it's immediate vicinity. It measures the centrality by counting the number of direct connections to the node. For a network with g nodes, the degree centrality of node n is defined as

$$C_D(n) = \frac{k(n)}{g-1}, \tag{1}$$

where $k(n)$ denotes the degree of node n. The Degree Centrality is normalized by $g-1$ since this is the highest possible degree, a node that is connected to every node in the network other than itself [Meghanathan, 2018].

2) Closeness Centrality: While the Degree Centrality only takes the adjacent nodes into account, it may be possible that the most central in this sense, i.e. the node with the highest degree, is in fact not close to other nodes in the network. This shortcoming is overcome by the *Closeness Centrality*. It considers the sum of geodesic distances, i.e. the number of edges in the shortest path between two nodes. The Closeness Centrality of a network with g nodes is defined as

$$C_C(n) = \frac{g-1}{\sum_{i=1}^{g} d(n,i)}, \tag{2}$$

where $d(i,j)$ stands for the geodesic distance between node i and j [Meghanathan, 2018].

3) Betweenness Centrality: So far, all the measures focus heavily on the direct influence of the node in question on others. However, two nodes which are not adjacent can also influence each other indirectly through other nodes in the network. The *Betweenness Centrality* highlights the nodes that lie most frequently in the connecting path between two nodes. For a network with g nodes the Betweenness Centrality for node n is defined by the sum over all geodesic distances that pass node n, normalized by the sum of all shortest paths of all pairs of nodes:

$$C_b(n) = \frac{\sum_{j<k} g_{jk}(n)}{g_{jk}}.$$

Here, g_{jk} denotes the number of shortest paths between two nodes of the netwok and $g_{jk}(n)$ marks the number of shortest paths that pass through node n. Since the maximum number of such paths is

$$\frac{(g-1)(g-2)}{2}.$$

One can thus further normalize:

$$C_B(n) = \frac{C_b(n)}{\frac{(g-1)(g-2)}{2}}. \tag{3}$$

4) Clustering Coefficient: In order to account for local structures in graph theory, one uses the *Clustering Coefficient*. There is evidence that suggests that the ties between nodes in most real-world networks tend to create tightly knit groups

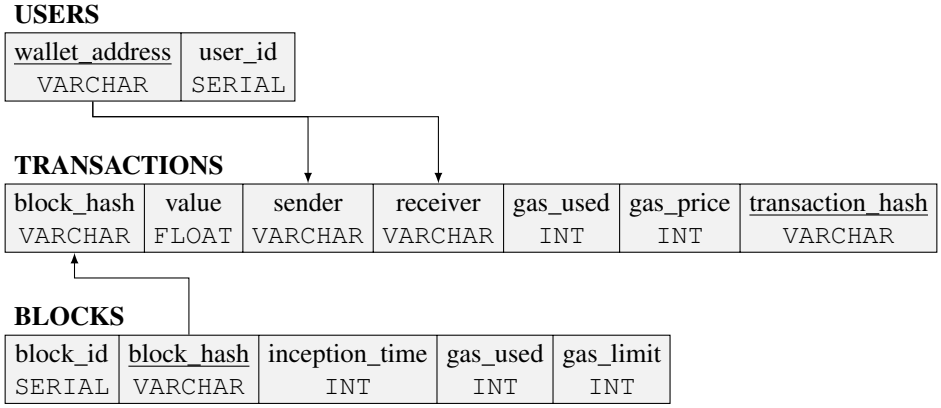


Fig. 9: Database Model: the primary keys of each table are underlined, foreign keys are represented by arrows, the type of each column is capitalized.

with a relatively high density of connections [Watts and H. Strogatz, 1998].

There exist two versions of this measure, the *global* and the *local* Clustering Coefficient. The global Clustering Coefficient C_G is the number of triangles over the total number of open or closed triplets in the graph. This measure can be applied to both directed and undirected networks.

The local Clustering Coefficient for a node n is given by the proportion of links between all adjacent vertices divided by the number of links that could possibly occur between them. For a directed graph the in- and outgoing edges have a different meaning and therefore the maximum number of immediately adjacent neighbors of n is $k(k-1)$, where k is the degree of node n. The neighborhood of a node n, i.e. its immediately connected neighbors, is defined as

$$N = \{n : e_{ij} \in E \cap e_{ji} \in E\},$$

where E is the set of edges.

Thus the local clustering coefficient of of node n is given by:

$$\begin{aligned} C_L(n) &= \frac{\text{number of closed triplets centered around n}}{\text{number of triplets centered around n}} \\ &= \frac{|\{e_{jk} : v_j, v_k \in N \cap e_{jk} \in E\}|}{k(k-1)}. \end{aligned} \tag{4}$$

In an undirected graph the meaning of in- and outgoing edges is considered identical. Therefore if a node n has degree k, there could exist a total of $\frac{k(k-1)}{2}$ edges within n's neighborhood. Thus the local Clustering Coefficient of a undirected network can be defined as

$$C_L(n) = \frac{2 \cdot |\{e_{jk} : v_j, v_k \in N \cap e_{jk} \in E\}|}{k(k-1)}. \tag{5}$$

This means that nodes with a high Clustering Coefficient play a large role when relaying information through the network [Knorr, 2019].

5) Power Law and the Small World Phenomenon: The small world phenomenon describes that the size of a network does not have any effect on the size of ties among its nodes. This means that the path lengths are characteristically small just like random graphs [Watts and H. Strogatz, 1998].

In a random graph the degree of a node n follows a power law distribution, i.e. we can say:

$$P\left(K(n) = k\right) \propto k^{-\alpha}, \tag{6}$$

where $K(n)$ is the random variable that describes the degree of node n and α is a constant whose values range between 1.6 and 3.0 [Newman, 2003]. This distribution suggests that most nodes will have a low degree, while a small number of nodes will display a large number of connections [Alstott et al., 2014].

VI. Findings

A. Data Preparation

We encountered many difficulties during our attempts at collecting and refining the data. At first, we attempted to deploy the data-scraper on a `RaspberryPi 3` Model B in conjecture with an external HDD hard drive. We decided on this option, since it did not require any of us to commit indispensable computing ressources and at that time it seemed to be the most convenient option.

That attempt was, in hindsight, doomed to fail due to the hardware's limitations. The problem did not lie in the hardware intensity of the data-scraper in itself, but rather in the inordinate amounts of memory it takes for `go-ethereum` to synchronize with the Ethereum blockchain. The interface to the ethereum blockchain, `go-ethereum`, is written in a manner that is primarily optimized for mining ether and in order to facilitate the commissioning of transactions and smart-contracts. The way the programm achieves this is by consuming as much memory as there is available on the system, or at least as much memory as the user is willing to allocate to it before starting the program. However, whenever the program runs out of memory or whenever it attempts to

exceed its predetermined limit the process shuts down and kills itself. This does not result in a system crash, but it does inhibit the workings of the `Python` libraries that are used inside the data-scraper.

This design of the `go-ethereum` program frequently led to the `RaspberryPi`'s 1 GB of memory being filled after only a short period of time, usually between thirty minutes and three hours. Note, that the data-scraper's construction did anticipate interruptions and it is possible to resume scraping the Ethereum blockchain from the current block without the loss of any data and without the need to redundandly query more than the one block at which the disconnection happened. This behaviour meant that the `RaspberryPi` was idling for most of the time.

Unfortunate as this was, it was not the only problem. The use of an external hard disk drive for storing the database meant that a considerable amount of the time during which the `RaspberryPi` was productive was spent on input/output-operations while it was commiting the scraped information into the database. In order to alleviate the bottleneck that was presented by the I/O-operations it would have been favorable to use an external solid state drive for that task. However, we were too frugal to obtain one just for the singular purpose of this seminar paper.

This means that after running the data-scraper for an entire month, along with taking all the care that is reasonable, in order to keep the scraper afloat, we were able to extract 120.000 transactions from 15.000 blocks which tantamounts to merely three days of data.

We solved all these problems by taking up the offer *Microsoft Azure* provides for students. Using this free initial student credit on the *Microsoft Azure* servers it was possible to set up a virtual machine with 32 GiB of memory along with access to an additional solid state drive. The fact that it is possible to rent out a `Linux` machine on the *Microsoft Azure* servers made it possible to migrate the scripts for the data-scraper without the need for adjustments.

TABLE I: Summary statistics

	value	gas used	gas price
mean	48.12	127511.76	3.01e+10
std	1495.49	239571.52	1.34e+13
min	0.00	21000.00	0.00
25%	0.06	40000.00	2.00e+10
50%	0.89	90000.00	2.00e+10
75%	1.11	90000.00	2.50e+10
max	1000000.00	4712388.00	3.62e+16

The initial student credit was enough to deploying the data-scraper on the *Microsoft Azure* servers for about a week. During this week we were able to extract 2 GB of data that consist of one million blocks with a total of 7.3 million transactions made from 415000 distinct wallets. Table I describes some rudimentary statistics.

Initially we also planned to scrape the IP-addresses of a wallet by making use of `web3.py`'s functionality which is provided for the `Bitcoin` blockchain. However, within the scraped dataset we were not able to identify a single address. The reason might that the `web3.py` module does not provide that functionality for the Ethereum blockchain in its current version. Mind you that we do not claim that it is impossible to retrieve the IP-address of a wallet. One proposed heuristic to do so is to assume that the first node that propagates a transaction through the network is also the one that is initializing the transaction [Reid and Harrigan, 2013]. However, for us this would have meant to rewrite the data-scraper from `Python` into the `Go` language, which we considered to be out of this analysis' scope.

B. Descriptive Analytics

Descriptive analysis of the data allows getting insights that can be useful at the time of the network analysis. By itself alone it can point at important information with regard to the network structure, interactions between the nodes, underlying business processes, design problems, potential privacy and security issues.

For instance, while the median value of a transaction is equal to 0,9 ether, the value of the biggest one is 1 million ether (138 million U.S. dollars as of 15.03.2019). Figure 10 shows the distribution of value sent in individual transactions. There is a significant number of outliers indicating at certain concentration of ether holdings.

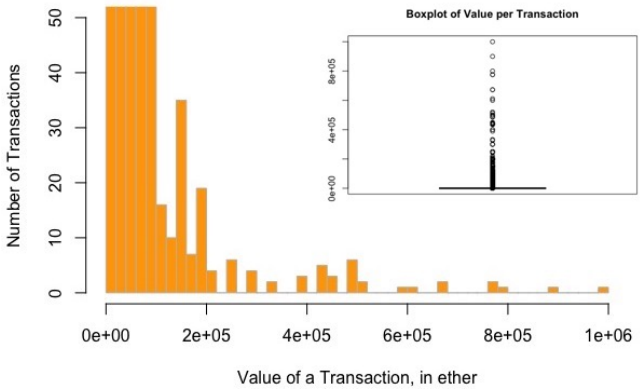


Fig. 10: Distribution of Value sent per Transaction.
**The number of transactions on the Y axis is limited to 50, so that the less frequent values remain visible.*

The amount of fees paid in transactions gives a hint at the underlying variation in businesses activity and Ethereum design issues. Cryptocurrencies are often promoted as a fast medium for micro-payments. However, fast transactions with low fees are not always the case. Limitations in the block size and the number of blocks in a period of time may result in an overflow of transactions pending to be included in a block. To speed up transactions people pay higher fees. At the same time, when the exchange rate of a cryptocurrecny rises, fees tend to become lower.

Interestingly, the is a big outlier - once there was a fee of more than 600 ether (80 000 U.S. dollars as of 15.03.2019). Apparently this is due to a mistake made by a payer. Such

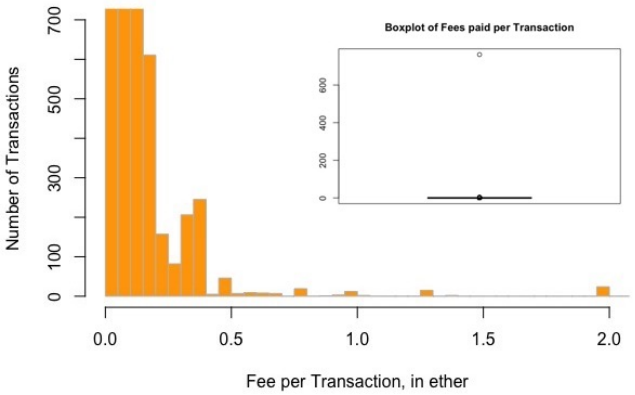


Fig. 11: Distribution of Fees paid per Transaction.
**The number of transactions on the Y axis is limited to 700, so that the less frequent fees remain visible.*

a mistake shows one of the disadvantages of the blockchain-based currencies where all transactions are irreversible.

Analysis of the distribution of gas used in transactions (see Figure 12) points at differences in their types. As gas measures how much "work" an action or a set of actions takes to perform, this potentially can be used to make a preliminary judgement about the prevalence of the smart contracts or about distribution of their complexity.

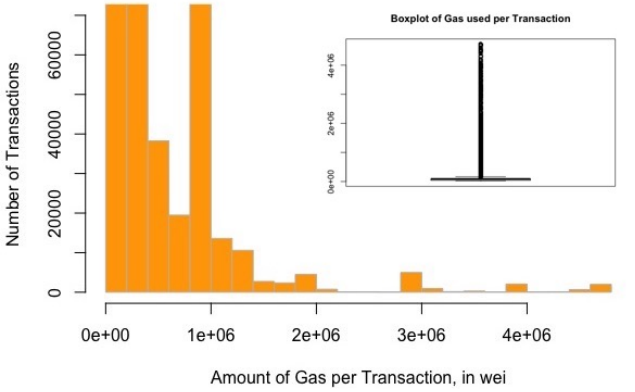


Fig. 12: Distribution of Gas used per Transaction.
**The number of transactions on the Y axis is limited to 70 000, so that the less frequent amount of gas remain visible.*

There are obviously two major clusters that probably separate regular payment transactions and smart contracts plus several groups of more complex contracts that however represent minority in the total volume of transactions. Both median and the 3rd quartile amount of gas is equal to 90 000 wei. In case the above hypothesis is correct, regular payment transactions will cover at least 75 % of the total number and are all represented on the Figure 12 by the 1st bar (cut in order to make other values visible).

Analysis of the transactions number over time makes the concern about unequal distribution of wealth in the network visible. As illustrated on the Figure 13 the number of transactions on the most active days more than doubles the respective number on the days with the lowest activity. Such significant variation indicates at the presence of major nodes.

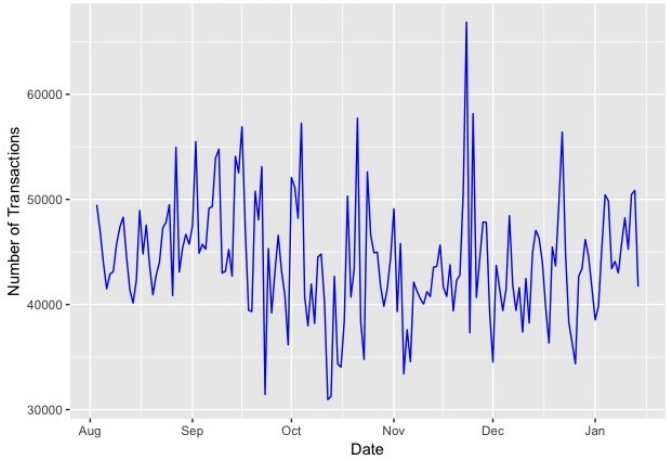


Fig. 13: Number of Transactions over Time.

This becomes even more visible if we consider the monetary volume of transactions over time (see Figure 14). Increases in volume of up to a hundred times from one day to another are an example of activity that would lead to a drastic changes in prices if these transactions are made not on chain but on currency exchanges.

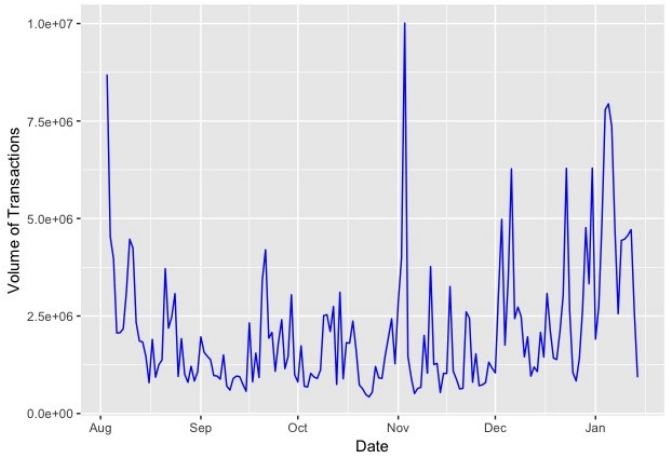
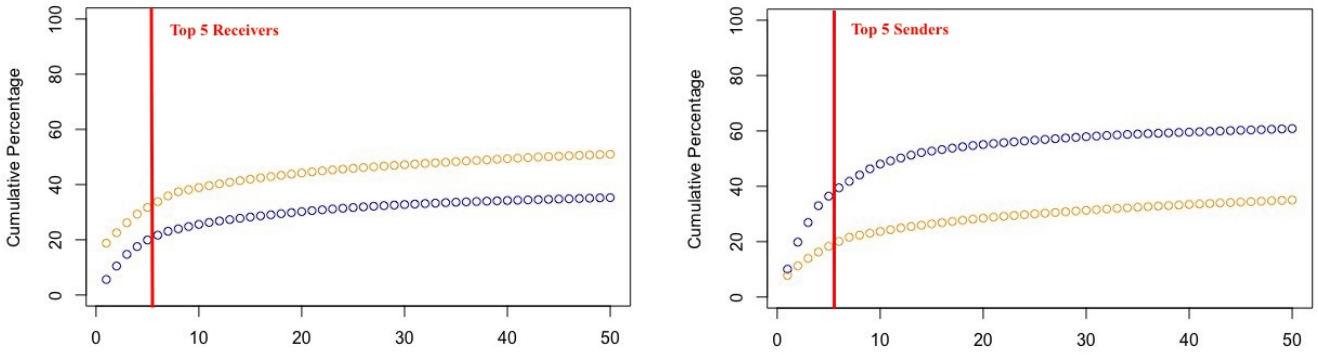


Fig. 14: Volume of Transactions over Time, in ether.

In order to verify the hypothesis about the presence of major nodes we analyse transactions number and volume more in depth. This is done by associating transactions with individual accounts. We treat receivers and senders of transactions separately.

As shown on the Figure 15a top 5 major accounts have received 19,9 % of all transactions (1,45 out of 7,3 million transactions). Top 50 accounts cover slightly more than one third of the total transactions' number. When it comes to the monetary value, the situation is even more centralized. Top 5 accounts have received 31,8 % of the total transactions' value



(a) Top 50 Receivers.

(b) Top 50 Senders.

Fig. 15: Cumulative Percentage of Transactions (blue) and Volume (orange) covered by Major Receivers (a) and Senders (b).

(112 out of 352 million ether received in all transactions). Major 50 Receivers cover more than a half of the volume transacted within the network.

Figure 15b shows the data for major senders of transactions. Here, in terms of the transactions' number the situation is even more centralized. Top 5 accounts have made 36,4 % of all transactions. Top 50 senders cover 60,1 % of the entire transactions' number. At the same time in terms of volume the picture is different. Top 5 and Top 50 senders cover only 18,3 % and 35,0 % of the total transactions' value respectively.

Interestingly, there are significant discrepancies in the degree of centralization if the data about receivers and spenders is compared to each other. This is the case for centralization both in terms of transactions' number and volume. Analysis of these discrepancies may shade light on some of the roles that major nodes play in the network.

Discrepancies in the Transactions' Volume between Receivers and Senders. There is a significant difference between the amount of ether received and spent by the major accounts. While Top 50 Receivers have accumulated 172 million ether, Top 50 Senders have spent only 123 million. Besides, top receivers and senders should not necessarily be the same entities. Consequently there is a number of major nodes that spend less than their receive or don't spend at all. Figure 16 shows the net balances of the accounts within the studied period of time. While absolute majority of accounts have balance around 0, there are few that have accumulated a significant amount of wealth.

Those could be mining pools that receive new coins and hold them anticipating appreciation of ether. However the amount of Ethereum accumulated by such nodes seems to be too big to be explained this way. Despite a new block in Ethereum is mined every 10-20 seconds and remunerates miners with 5 ether, during the analyzed period of time this results in only approximately 5 million new ether.

Probably, these nodes are currency exchanges that accumulate wealth of their participants. They keep track of exchange transactions internally without recording them on-chain. The wealth is recorded as spent only when a participant makes a payment transaction.

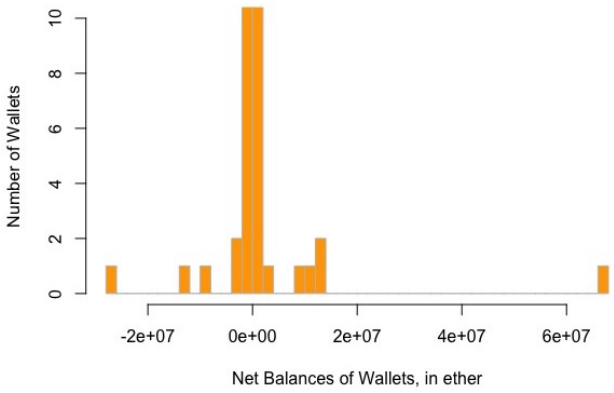


Fig. 16: Distribution of Balances within Studied Period. **The number of wallets on the Y axis is limited to 10, so that the less frequent wallets remain visible.*

If the nodes are actually currency exchanges, there is a deviation from the Ethereum narrative of "no trusted third party involvement". Given the level of centralization associated with the nodes, this could imply a major source of vulnerability. An attack on the exchange records would lead to losses for a significant number of the exchange participants.

In order to verify how relevant this risk actually is, we conducted further investigation of the accounts that received the biggest volumes of ether. Using data of the Ethereum scanners such as Etherscan.io, Etherchain.org, Blockchair.com, Bloxy.info, etc.[1] we found out affiliation of the Top 5 Receivers (in terms of volume).

Four out of five Top Receivers actually belong to the currency exchanges (see Table II). However, all these accounts in reality appeared to be not regular addresses but smart contracts. The remaining Top Receiver (the first one in the list) is also a smart contract. None of them is supposed to

[1]For details see:
https://etherscan.io/,
https://www.etherchain.org/,
https://blockchair.com/Ethereum/,
https://bloxy.info/.

TABLE II: Top 5 Major Receivers (Volume)

Wallet	Amount Received, mill. ether	Type	Affiliation
0xAA1A6e3e6EF20068f7F8d8C835d2D22fd5116444	66,0	Smart-Contract	ReplaySafeSplit
0xBFC39b6F805a9E40E77291afF27aeE3C96915BDD	13,2	Smart-Contract	Poloniex
0x209c4784AB1E8183Cf58cA33cb740efbF3FC18EF	12,6	Smart-Contract	Poloniex 2
0x7727E5113D1d161373623e5f49FD568B4F543a9E	11,1	Smart-Contract	Bitfinex 2
0xFa52274DD61E1643d2205169732f29114BC240b3	8,5	Smart-Contract	Kraken 4

TABLE III: Top 5 Major Senders (Number of TXs)

Wallet	Number of TXs send	Type	Affiliation
0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8	733818	Address	Ethermine
0x2a65Aca4D5fC5B5C859090a6c34d164135398226	715017	Address	DwarfPool 1
0xD34DA389374CAAD1A048FBDC4569AAE33fD5a375	516932	Address	GenesisMining
0x52bc44d5378309EE2abF1539BF71dE1b7d7bE3b5	443544	Address	Nanopool
0x61C808D82A3Ac53231750daDc13c777b59310bD9	251566	Address	F2Pool 1

accumulate wealth and by design should transfer it further. This puts under question the validity or completeness of the transactions data we have collected.

Deeper research allowed restoring the faith in the dataset. Appeared that money received by identified major accounts was actually spent. Bit in another network.

The point is that on the 20th of July 2016 (shortly before the analyzed period) Ethereum experienced a hard fork. As a consequence of the DAO incident - a hack of a complicated smart-contract that resulted in a loss of about 12.7 million ether (worth around 150 million U.S. dollars at the time) - the community was split in two parts. Most of the network participants decided to make a fork chain where the stolen coins would be returned to their owners. However, some of the community decided to continue the old chain (Ethereum Classic or ETC), arguing that "the code is the law" and the blockchain data should be irreversible[2].

This lead to a number of issues related to the interaction between chains. One of the most relevant of them was "replay attack". The mechanics of it is as follows: if there is a valid transaction on one chain, it can also be offered - replayed - on the other chain to duplicate the received amount. For instance, a user of the currency exchange can deposit and withdraw her money from this exchange on the old chain, and then use the 2nd transaction to withdraw money also on the new (forked) chain[3]. This concern was solved through inter-mediation of the smart contracts.

Thus, the post by Ethereum founder Vitalik Buterin states: "users who are interested in taking any actions with their ETC, including creating and participating in applications, converting to another asset, etc. are advised to use the splitter contract at address 0xAA1A6e3e6EF20068f7F8d8C835d2D22fd5116444 to move their ETC to a separate newly created account so as to avoid replay attacks; we also encourage the ETC community to consider adopting a secondary hard fork to change transaction formats to make further replay attacks impossible[4].

The address mentioned in the post is the biggest Receiver in our dataset[5]. It has transmitted all the received money to its senders, but on the other chain. The other 4 accounts, which are intermediate exchange wallets were apparently performing similar function of splitting wealth between different chains.

Thus, our concern about possible attack is confirmed only partially. None of the Top 5 Receivers were accumulating wealth and it all was automatically transmitted to other accounts. At the same time while exploring the "ReplaySafeSplit" smart-contract we have found that a number of users lost their money due to the mistakes made by them during the split[6]. The smart-contract itself appeared to be susceptible to a number of bugs[7] and was later substituted by another one. The DAO incident described in the context of our Receivers exploration also exemplifies how the risk of the attack on an account or contract that holds funds from different users can and has actually played out.

Discrepancies in the Transactions' Number between Receivers and Senders. As has been described by the Figure 15 the number of transactions received by major Receivers is significantly smaller than the number of transactions made by major Senders. This discrepancy can probably be partially explained by the activity of the mining pools. There, one account receives a mining reward and then sends it in small fractions to the many participants of the pool.

Alternative explanation can be related to various approaches of the network participants to gain more privacy. One of them is the so called "peeling-chain". The technique consists in dividing and sending the wealth of an account to multiple addresses again and again. The goal is to create an impression that several users are doing transactions instead of one.

[2]See for instance https://www.cryptocompare.com/coins/guides/the-dao-the-hack-the-soft-fork-and-the-hard-fork/.

[3]See for instance https://vessenes.com/hard-fork-and-replay-concerns/.

[4]For details see https://blog.Ethereum.org/2016/07/26/onward_from_the_hard_fork/.

[5]The original address in the post was substituted as the original smart-contract was changed.

[6]See for instance https://medium.com/@chevdor/safer-version-of-the-replaysafesplit-smart-contract-a29c347e8a7.

[7]For details see https://etherscan.io/address/0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444#code.

The process is illustrated on the Figure 17. Here, the received amount of 50 coins is splitted in several iterations to multiple accounts in an attempt to complicate tracking of a certain persons wealth.

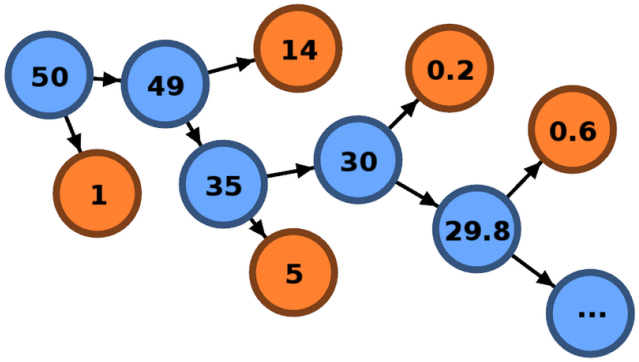


Fig. 17: How the peeling-chain works. From [de Balthasar and Hernandez-Castro, 2017]

Further research of the Top 5 transaction Senders allowed verifying the hypotheses about the role of different nodes and their activities. Using data of the Ethereum scanners as before we disclosed identities of the major accounts. All of them appeared to be mining pools (see Table III. Among Top 5 Receivers of transactions there are no mining pools (the accounts are a digital token YoCoin, 2 Poloniex exchange wallets, currently closed by the U.S. government cryptocurrency trading platform BTC-e and already mentioned ReplaySafeSplit smart-contract). Thus the hypothesis about the activity of mining pools is confirmed. "Peeling-chain" practices are neither confirmed nor disproved.

C. Graph Analysis

Looking at the rough outline of the observed period revealed that almost half of the wallets take part in one transaction during that time frame. Roughly 16 percent of the wallets appear two or three times respectively.

Since the vast majority of accounts only seem to appear once inside the whole dataset we concluded that these are not interesting from a graph analysis perspective. Therefore we opted to use a subset from that data of accounts, where we could verify that they are interacting with other accounts. This subset consists of roughly four and a half million transactions in which 27.000 distinct wallets were either the sender or the receiver. From this subset we opted to create a graph in which the wallets are represented by the graph's nodes and the transactions are symbolized by the graph's edges.

Even this subset proved to be too large to handle, which means for the further graph-analysis that we were forced to rely on a random subset. However, if this random sample is large enough it should still give us an insight into the structure of the Ethereum network and the graph formed by this subset should still provide interpretable results.

Figure 19 displays the graph built from a random subsample of 1000 transactions, where again the nodes represent the different wallets and the edges stand for an observed transaction between them. The size of the node stands for the transaction's value. We can clearly see that most of the accounts form connections that consist of only two wallets and it is highly likely that these are also the accounts which are only observed once in the entire dataset.

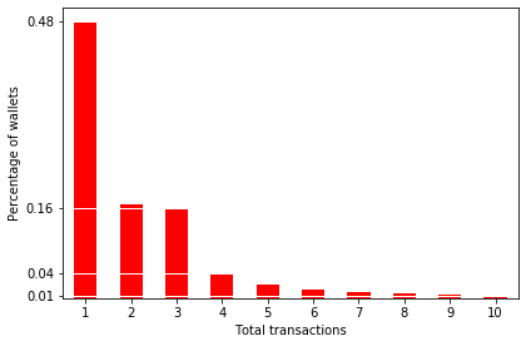
Even here it is possible to see that there are subclusters of highly connected nodes associated with a high transaction value located in the middle of the graph. This appears despite the small sample size, which has to mean that these nodes in the center of the graph are important actors inside the network whose position will most likely become more pronounced the larger we choose the sample size to be. From this first glimpse at the network we would expect to find two tendencies inside the dataset. First, we expect a large number of nodes which are associated with low centrality measures and clustering coefficients. Secondly we expect to find some collections of nodes that form subgroups and within these subgroups we should be able to locate some nodes that play a tangible role in their respective local neighborhoods. Both of those observations do fit into the notion of a random graph, where we would expect a large number of loosely connected nodes with a low degree and an exponentially lower quanity of nodes which have a higher degree and who play a more central role inside the network.

Note nonetheless, that figure 19 provides only a rough outline since it is only using one thousand out of the 7.3 million available transactions. We pick a comparatively small sample size for this graph in order to avoid overplotting, the further analysis however uses a larger subsample of 100000 transactions.

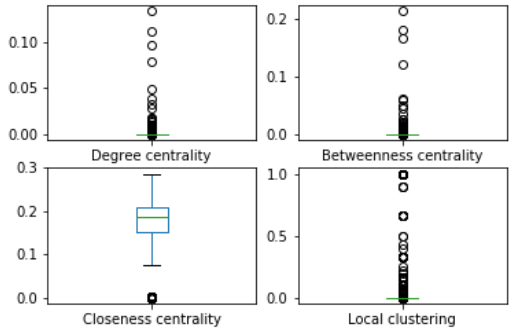
The results of the larger subsample are summarized in table IV and figure 18b. For the Degree Centrality we can observe a highly skewed distribution, where the mean lies at $\bar{C_D} = 0.0001$, whereas the first three quartiles remain firmely at 0. This indeed confirmes our previous suspicion that most nodes inside this network are merely loosely connected, even after removing the most isolated and remote nodes of the initial dataset. This means that most wallets interact only with are very limited set of other wallets and that it is highly uncommon for a node to accept transactions from many different wallets.

When looking at the results for the graph's Betweenness-Centrality, we can see a similiar pattern. The mean lies at $\bar{C_B} = 0.0001$ and all three quartiles are, again, 0. This means that the Betweenness centrality's distribution is also heavily skewed, which indicates that only a few nodes act as a mediator for their neighbors. This should not come as a surprise, since most of the nodes have a very low degree to begin with.

The Closeness centrality between all nodes paints a different picture. Its distribution is still skewed, however it is not as extremely skewed as the measures that have been discussed so far. The mean lies at $\bar{C_C} = 0.15$ with a standard deviation of $s_{C_C} = 0.078$. Note that this measure needs to be interpreted



(a) Distribution of total number of transactions



(b) Centrality Distribution

TABLE IV: Centrality Distribution

	Degree centrality	Betweenness centrality	Closeness centrality	Local clustering
mean	0.0001	0.0001	0.1540	0.0043
std	0.0012	0.0020	0.0782	0.0636
min	0.0000	0.0000	0.0000	0.0000
25%	0.0000	0.0000	0.1522	0.0000
50%	0.0000	0.0000	0.1876	0.0000
75%	0.0000	0.0000	0.2072	0.0000
max	0.1344	0.2149	0.2862	1.0000

with care. Most of the nodes are disconnected, and thus do not have a connecting path at all. If a node j cannot be reached from a different node k there exist two common propositions in the literature. Either $d(j,k) = 0$ or $d(j,k) = \infty$. Most implementations rely on the former convention rather than the latter, since it avoids values that cannot interpreted as a number. This is also the convention that is implemented in the `networkx` module for the `Python` programming language [Hagberg et al., 2008]. But this also means that the closeness centrality should be interpreted as a local measure since the distances in the normalizing factor in (2), which are calculated as

$$\sum_{i=1}^{g} d(n,i),$$

will contain numerous zero-valued summands. So actually, this measure only makes sense for nodes that are of a degree k that is larger than three, since nodes of a lesser degree are necessarily close to each other.

The Local clustering coefficient accross the nodes is again highly skewed to the left. The mean lies at $\bar{C_L} = 0.004$ at a standard deviation of $s_{C_L} = 0.063$. Again the quartiles are all planted at 0. Note however, that there exist some nodes whose Local clustering coefficient reaches as far as $C_L(n) = 1$. This means that there are some nodes who are fully connected within their local cluster, which means that all triangles within that clique are centered around that node n. This, in turn, means that there are some nodes which are perfectly connected inside their neighborhood.

So far all these findings indicate that the model of a random graph would be a good approximation of the real world data. In order to test this assumption in a graphical manner, we would like to compare the empirical density functions of the degrees against the theoretical powerlaw distribution. Remember that, in a random graph, the random variable $K(n)$ that describes the degree of a node n follows a powerlaw distribution

$$P(K(n) = k) \propto k^{-\alpha},$$

where α is some constant that has to be determined by Maximum-Likelihood methods [Meghanathan, 2018]. Figure 20 presents the results for the empirical probability density function (PDF) and the complementary cumulative distribution function (CCDF). The CCDF is primarily used in the domain of signal processing and measures the the amount of time a signal spends above the mean power level of the measured signal. Equivalently, it assesses the probability that the signal power will be above the average power level [Udaigiriya and Sharma, 2015].

In order to reflect the difference in the roles accross the network such as the reciever or the sender of a transaction, we split the analysis into three different parts. First, we look at the distribution of the directed graph that is focusing on the actors that iniatate a transactions, i.e. the senders. Second, we look at the network formed by the receivers of a transaction and finally we look at the whole undirected graph that contains both receivers and senders. The results are displayed in figure 20. The PDF is shown by the blue line and the CCDF is represented by the red line. A high overlap between the dashed and the solid line indicates a good fit of the empirical distribution compared to the theoretical. Thus the probability density function of a perfectly random network would lie on its respective dashed line.

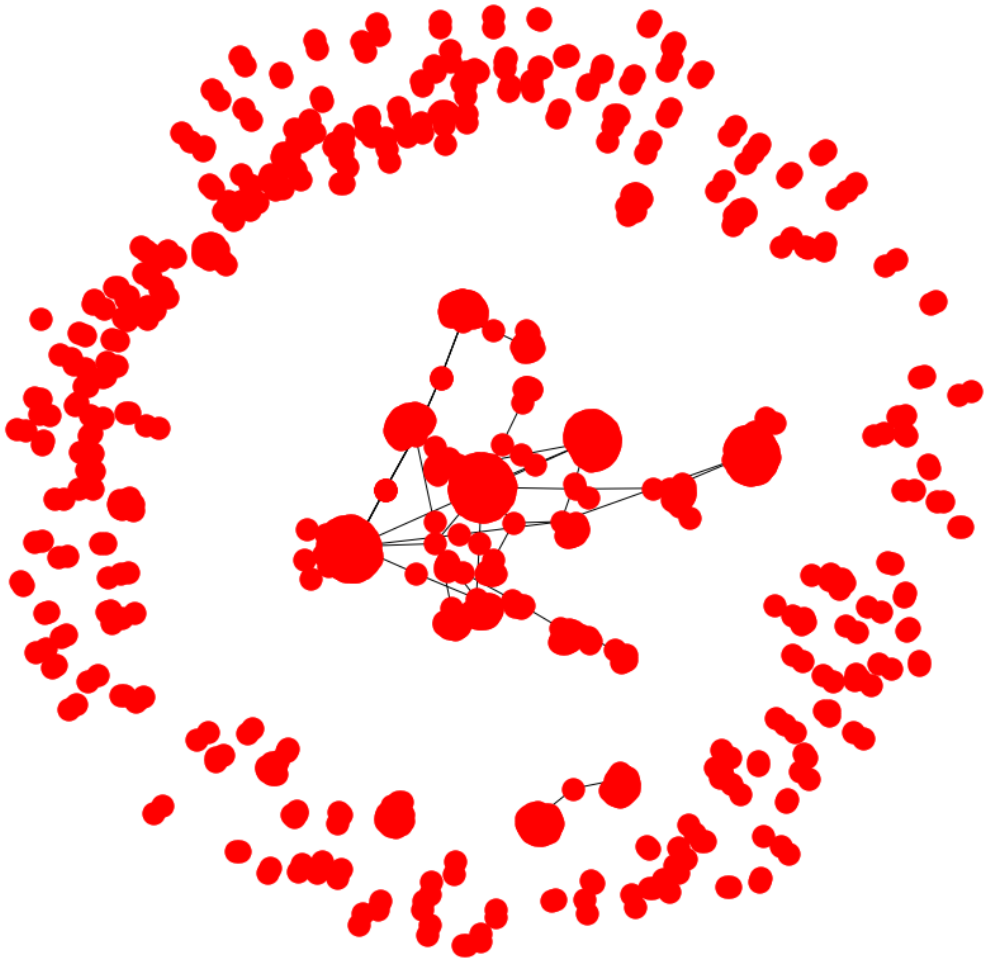


Fig. 19: Graph formed by a subsample of 1000 randomly drawn wallets

In the leftmost part of figure 20 we can see for the senders that the tails of the empirical PDF are closely following the theoretical distribution of the random graph. However, in the central part we observe one large deviation from the distribution of a random network. In particular, this means that - under a random graph - we would have expected more nodes that have a middling degree, i.e. $k = 3$, $k = 4$, and so forth. Instead, we observe a large number of loosely connected nodes.

The same holds for the sender's CCDF. The lower tail of the distribution follows the random graph closely, whereas the central part is too low, and the upper tail is too high for a random graph.

Looking at the wallets at the receiving end of a transaction, in the middle part of figure 20, we can discern a similar pattern. Both, the line for the PDF, as well as the line for the CCDF begin by closely following the theoretical distribution and begin to deviate after passing the median. Again we observe a

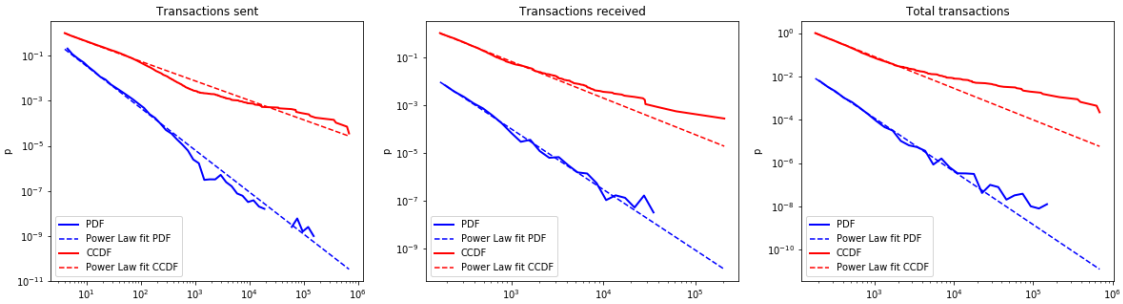


Fig. 20: Comparison with random graphs

density at the lower tail that is very similar to what we would expect from a random graph.

However, the most clear indicator of that pattern is shown for the composite graph, that contains both senders and receivers, in the rightmost part of figure 20. Here we observe a very pronounced deviation from the course of both the theoretical PDF and CCDF after reaching the median. Again, this is a clear indicator that there are too many nodes with a comparatively high degree, for this to be a random network.

VII. Conclusion

This work wanted to transfer the approaches of the Bitcoin paper [Lischke and Fabian, 2016] to the Ethereum blockchain and get some insights into the Ethereum Economy. To do so, transaction data got scraped, displayed and analyzed.

Beforehand we looked into studies already done, to get a better understanding what was done so far and might find some more information about how to get good off set data or IP addresses. As the Ethereum is still quite new, there are some paper out there, who already looked into the Ethereum address spaces, security aspects and theoretical aspects of the Ethereum technology to find patterns within the network. None of them brought together transaction data and network data to perform an analysis to find pattern if they are getting combined with offset network data over a longer period.

With that knowledge the author worked his way through the same platforms, bitcoin.info to enrich our data set and analyze it in the descriptive and graph analysis. Even if we were not able to get data about the IP addresses, the descriptive analysis could deliver information to get related to the network analysis, network structure, interactions between the nodes and underlying business processes. Also design problems can be found and potentiation privacy and security issues.

Network structure Interactions between the nodes Underlying business processes Design problems Potential privacy and security issues

To find some major nodes in the Ethereum network, the centrality distribution was performed on the data. Major results are, that within the network one can find major hubs, with high transaction volume. Most of the nodes are not connected. They are usually just having little transactions between each other. The power law shows, that the data set taken into consideration is fitting the power law contribution.

Originally the Ethereum protocol was thought of as a modified version of the Bitcoin Blockchain. Thats what Vitalik Buterin had in mind when he wrote the white paper in 2014. The turing completeness should provide opportunities to create applications for any type of application [Buterik, 2013] Vitaliks idea was to get optimize the Etherum Blockchain technology to make the blockchain technology more applicable for a bigger amount of applications. He thought Ethereum being open ended by design and believes that it is extremely well suited to serve as a basic layer for financial and non-financial applications. [Buterik, 2013]

As shown in the comparison between blockchain and bitcoin, one could guess, that so far this goal is not close to be reached by the Ethereum community. There are lots of attempts to create applications for the platform and try to find business cases. Indicator is, that the number of active used Ethereum addresses has been dropped below 1 million and also that a study on usage of smart contracts indicated, that 95

To conclude, we were not able to perform such a comprehensive study as it was done from Lischke and Fabian and the insights into the Ethereum transactions and regarding information are less than expected. We had some limitations along the way. Crucial was the access to the different data sets and the amount of data we were able to collect. The data were uncompleted, and we were not able to locate the transactions via IP addresses. In the replicated paper they were able to scrape the data with the transaction data, using blockchain.info. This is not the possible for the Ethereum and a proper workaround was not easily found. We searched for alternative providers, but none of them gave actually information about the IP address were the transaction is coming from. Another challenge her is, that the network different from the Bitcoin also provides smart contracts and a currency service, here the distinction is not easily made in the data. Some use the platform just for transactions and a lot of them apply smart contracts or decentralized applications.

Because we could not perform the step of gathering IP addresses, the economic tags did not even get a relevance. As mentioned in the methods, for a further paper one could look

in more detail into the scraper we applied and modify it for the usage in the go language. To get back to the data, unfortunate the time period gathered was during a time when a hard fork happened in the Ethereum network, in 2017. All that happened because of a hard fork incidence, when a complicated hack of a smart contracts leaded to a loss of 12,7 million US Dollar. The network got divided and had to reorganize. This brought noise in the data, which is not the usual situation and to get broader explanations, one should take care to use a different period or a longer one, to get more general insights.

REFERENCES

[Alstott et al., 2014] Alstott, J., Bullmore, E., and Plenz, D. (2014). powerlaw: A python package for analysis of heavy-tailed distributions. *PloS one*, 9:e85777.

[Anderson et al., 2016] Anderson, L., Holz, R., Ponomarev, A., Rimba, P., and Weber, I. (2016). New kids on the block: an analysis of modern blockchains. *arXiv preprint arXiv:1606.06530*.

[Androulaki et al., 2013] Androulaki, E., Karame, G. O., Roeschlin, M., Scherer, T., and Capkun, S. (2013). Evaluating user privacy in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 34–51. Springer.

[Anoaica and Levard, 2018] Anoaica, A. and Levard, H. (2018). Quantitative description of internal activity on the ethereum public blockchain. In *New Technologies, Mobility and Security (NTMS), 2018 9th IFIP International Conference on*, pages 1–5. IEEE.

[Bartoletti et al., 2017] Bartoletti, M., Carta, S., Cimoli, T., and Saia, R. (2017). Dissecting ponzi schemes on ethereum: identification, analysis, and impact. *arXiv preprint arXiv:1703.03779*.

[Baumann et al., 2014] Baumann, A., Fabian, B., and Lischke, M. (2014). Exploring the bitcoin network. volume 1.

[BitInfoCharts, 2019] BitInfoCharts (2015-2019). Ethereum (eth) price stats and information. [Online; accessed 10-03-2019].

[Buterik, 2013] Buterik, V. (2013). A next generation smart contract & decentralized application platform. [Online; accessed 2019-03-21].

[Buterin et al., 2014] Buterin, V. et al. (2014). A next-generation smart contract and decentralized application platform. *white paper*.

[Butin, 2015] Butin, V. (2015). Merkling in ethereum. online, access 14 Mar 2019.

[Chan and Olmsted, 2017] Chan, W. and Olmsted, A. (2017). Ethereum transaction graph analysis. In *2017 12th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 498–500. IEEE.

[Chen et al., 2018] Chen, T., Zhu, Y., Li, Z., Chen, J., Li, X., Luo, X., Lin, X., and Zhange, X. (2018). Understanding ethereum via graph analysis. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 1484–1492. IEEE.

[de Balthasar and Hernandez-Castro, 2017] de Balthasar, T. and Hernandez-Castro, J. (2017). An analysis of bitcoin laundry services.

[Drainville, 2012] Drainville, D. (2012). An analysis of the bitcoin electronic cash system. *Waterloo, Canada: University of Waterloo*.

[Grishchenko et al., 2018] Grishchenko, I., Maffei, M., and Schneidewind, C. (2018). A semantic framework for the security analysis of ethereum smart contracts. In *International Conference on Principles of Security and Trust*, pages 243–269. Springer.

[Hagberg et al., 2008] Hagberg, A. A., Schult, D. A., and Swart, P. J. (2008). Exploring network structure, dynamics, and function using networkx. In Varoquaux, G., Vaught, T., and Millman, J., editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA.

[Kaminsky, 2011] Kaminsky, D. (2011). Black ops of tcp/ip. *Black Hat USA*, page 44.

[Knorr, 2019] Knorr, C. (2019). *Holland/Leinhardt (1971): Transitivity in Structural Models of Small Groups*, pages 267–270.

[Li et al., 2017] Li, Y., Zheng, K., Yan, Y., Liu, Q., and Zhou, X. (2017). Etherql: a query layer for blockchain system. In *International Conference on Database Systems for Advanced Applications*, pages 556–567. Springer.

[Lischke and Fabian, 2016] Lischke, M. and Fabian, B. (2016). Analyzing the bitcoin network: The first four years. *Future Internet*, 8(1):7.

[Maesa, 2018] Maesa, D. D. F. (2018). Blockchain applications: Bitcoin and beyond.

[Meghanathan, 2018] Meghanathan, N. (2018). Graph theoretic approaches for analyzing large-scale social networks. pages 7–34. IGI Global.

[Meiklejohn et al., 2013] Meiklejohn, S., Pomarole, M., Jordan, G., Levchenko, K., McCoy, D., Voelker, G. M., and Savage, S. (2013). A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 127–140. ACM.

[Newman, 2003] Newman, M. E. J. (2003). The structure and function of complex networks. *SIAM Review*, 45:167–256.

[Ober et al., 2013] Ober, M., Katzenbeisser, S., and Hamacher, K. (2013). Structure and anonymity of the bitcoin transaction graph. *Future internet*, 5(2):237–250.

[Ortega, 2013] Ortega, M. S. (2013). *The bitcoin transaction graphanonymity*. PhD thesis, Masters thesis, Universitat Oberta de Catalunya.

[Partz,] Partz, H. Ethereum unique addresses break 50 million, active wallet number keeps dropping - cointelegraph. [Online; accessed 10-03-2019].

[Payette et al., 2017] Payette, J., Schwager, S., and Murphy, J. (2017). Characterizing the ethereum address space.

[Piper and Caver, 2018] Piper, M. and Caver, J. (2018). Web3py: A Python interface for interacting with the ethereum blockchain and ecosystem. [Online; accessed 10-03-2019].

[Portal, 2019a] Portal, S. T. S. (2017-2019a). Price of ethereum from february 2017 to february 2019 (in u.s. dollars). [Online; accessed 10-03-2019].

[Portal, 2019b] Portal, S. T. S. (2017-2019b). Size of the bitcoin blockchain from 2010 to 2019, by quarter (in megabytes). [Online; accessed 10-03-2019].

[Preethi, 2017] Preethi, K. (2017). [Online; accessed 2019-03-21].

[Ray, 2018] Ray, J. (2018). Proof-of-stake faq's. online, access 14 Mar 2019.

[Reid and Harrigan, 2013] Reid, F. and Harrigan, M. (2013). An analysis of anonymity in the bitcoin system. In *Security and privacy in social networks*, pages 197–223. Springer.

[Rossum, 1995] Rossum, G. (1995). Python reference manual. Technical report, Amsterdam, The Netherlands, The Netherlands.

[Rudlang, 2017] Rudlang, M. (2017). Comparative analysis of bitcoin and ethereum. Master's thesis, NTNU.

[Sapuric et al., 2017] Sapuric, S., Kokkinaki, A., and Georgiou, I. (2017). In which distributed ledger do we trust? a comparative analysis of cryptocurrencies. *MCIS 2017 Proceedings*.

[Somin et al., 2018] Somin, S., Gordon, G., and Altshuler, Y. (2018). Social signals in the ethereum trading network. *arXiv preprint arXiv:1805.12097*.

[Spagnuolo et al., 2014] Spagnuolo, M., Maggi, F., and Zanero, S. (2014). Bitiodine: Extracting intelligence from the bitcoin network. In *International Conference on Financial Cryptography and Data Security*, pages 457–468. Springer.

[Tikhomirov et al., 2018] Tikhomirov, S., Voskresenskaya, E., Ivanitskiy, I., Takhaviev, R., Marchenko, E., and Alexandrov, Y. (2018). Smartcheck: Static analysis of ethereum smart contracts.

[Udaigiriya and Sharma, 2015] Udaigiriya, L. and Sharma, S. K. (2015). Complementary cumulative distribution function for performanceanalysis of ofdm signals and implement papr reduction techniques in ofdm system usinng ccdf function on matlab.

[Watts and H. Strogatz, 1998] Watts, D. and H. Strogatz, S. (1998). Collective dynamics of small world networks. *Nature*, 393:440–2.

[Wood, 2014] Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32.

VIII. Appendix

A. Code - Datascraper

```
from configparser import ConfigParser


def config(filename="database.ini", section="postgresql"):
    parser = ConfigParser()
    parser.read(filename)

    db = {}
    if parser.has_section(section):
        params = parser.items(section)
        for param in params:
            db[param[0]] = param[1]
    else:
        msg = "Section {} not found in the {} file"
        raise Exception(msg.format(section, filename))

    return db


import os
import psycopg2
from config import config
from glob import glob


def create_tables(command):
    """Create tables for ethereum database."""
    conn = None
    try:
        params = config()
        conn = psycopg2.connect(**params)
        cur = conn.cursor()
        cur.execute(command)
        cur.close()
        # If you don't commit, no changes will be made.
        conn.commit()
    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
    finally:
        if conn is not None:
            conn.close()


if __name__ == "__main__":
    queries_path = os.path.join("sql", "create", "*")
    queries = glob(queries_path)
    # HACK: table ip-info depends on transactoins, sort to fix order.
    queries.sort(reverse=True)
    for query in queries:
        with open(query) as q:
            command = q.read()
            create_tables(command)
```

```
CREATE TABLE IF NOT EXISTS blocks (
  block_hash VARCHAR(255) PRIMARY KEY,
  gas_used INTEGER,
  gas_limit INTEGER,
  block_id INTEGER UNIQUE,
  inception_time INTEGER
)

CREATE TABLE  IF NOT EXISTS ip_info (
  transaction_hash VARCHAR(255) NOT NULL,
  ip_address VARCHAR(255) NOT NULL
)

CREATE TABLE IF NOT EXISTS  trade (
  trade_date INTEGER PRIMARY KEY,
  open FLOAT,
  high FLOAT,
  low FLOAT,
  close FLOAT,
  value_eth FLOAT,
  value_usd FLOAT
)

CREATE TABLE  IF NOT EXISTS transactions (
  transaction_hash VARCHAR(255) NOT NULL,
  block_hash VARCHAR(255) NOT NULL,
  sender VARCHAR(255),
  receiver VARCHAR(255),
  value FLOAT,
  gas_used INTEGER,
  gas_price BIGINT
)

CREATE TABLE  IF NOT EXISTS users (
  user_id SERIAL PRIMARY KEY,
  wallet_address VARCHAR(255) NOT NULL
)


import psycopg2
from config import config


def connect():
    """Connect to the PostgreSQL database server."""
    conn = None
    try:
        params = config()
        # Create connection.
        print("Connecting to the PostgreSQL database...")
        conn = psycopg2.connect(**params)
        # Create cursor.
        cur = conn.cursor()

        # Execute a statement
        cur.execute("SELECT version()")
```

```
        # Fetch results and display them.
        db_version = cur.fetchone()
        print("Database version:", db_version)

        # Close connections
        cur.close()

    except (Exception, psycopg2.DatabaseError) as error:
        print(error)

    finally:
        if conn is not None:
            conn.close()
            print("Database connection closed.")


if __name__ == "__main__":
    connect()
```

```
from configparser import ConfigParser


def config(filename="database.ini", section="postgresql"):
    parser = ConfigParser()
    parser.read(filename)

    db = {}
    if parser.has_section(section):
        params = parser.items(section)
        for param in params:
            db[param[0]] = param[1]
    else:
        msg = "Section {} not found in the {} file"
        raise Exception(msg.format(section, filename))

    return db
```

```
import os
import psycopg2
from config import config
from web3.auto import w3
from hexbytes import HexBytes
from blockchain import blockexplorer


def extract_info(dictlike, keys):
    """
    Extract values from a dictionary-style object.

    * Parameters:
        * `dictlike`: object that has all attributes in `keys`
        * `keys`: list of keys to loop over.

    * Returns:
        * `out`: tuple of values corresponding to each key in `keys`.
```

```
    """
    out = []

    for key in keys:
        val = getattr(dictlike, key, "NULL")
        if val is None:
            val = "NULL"
        if type(val) == HexBytes:
            out.append(val.hex())
        else:
            out.append(val)

    return tuple(out)


def update(dictlike, table, keys):
    """
    Update a database-table with the contents of one block.

    * Parameters:
        * `dictlike`: object that has a `__get__`-method implemented.
        * `table`: (`str`) name of table to update.
        * `keys`: list of keys to loop over.
    """

    row = extract_info(dictlike, keys)

    sql = "INSERT INTO {} VALUES {}".format(table, row)
    conn = None
    try:
        params = config()
        conn = psycopg2.connect(**params)
        cur = conn.cursor()
        cur.execute(sql)
        conn.commit()
        cur.close()
    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
    finally:
        if conn is not None:
            conn.close()


def update_list(block, table, keys, block_key="transactions"):
    """
    Batch update a database-table with the contents of one block.

    * Parameters:
        * `block`: object that has a `__get__`-method implemented.
        * `table`: (`str`) name of table to update.
        * `keys`: list of keys to loop over.
        * `block_key`: (`str`) key of `block` that contains an iterable.
    """
    conn = None
    try:
        params = config()
```

```
        conn = psycopg2.connect(**params)
        cur = conn.cursor()

        for transaction in block[block_key]:
            tx = w3.eth.getTransaction(transaction)

            info = extract_info(tx, keys)
            sql = "INSERT INTO {} VALUES {}".format(table, info)
            cur.execute(sql)
        conn.commit()
        cur.close()
    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
    finally:
        if conn is not None:
            conn.close()


def tx_iterator(block):
    for transaction in block.transactions:
        yield transaction.hex()


if __name__ == "__main__":
    block = w3.eth.getBlock(6753133)
    block_keys = ["hash", "gasUsed", "gasLimit", "number", "timestamp"]
    tx_keys = ["blockHash", "hash", "from", "to", "value", "gas", "gasPrice"]
    ip_keys = ["hash", "relayed_by"]

    update(block, 'blocks', block_keys)
    update_list(block, 'transactions', tx_keys)
    for tx in tx_iterator(block):
        try:
            transaction = blockexplorer.get_tx(tx)
            update(transaction, 'ip_info', ip_keys)
        except Exception as e:
            print(e)
            continue


def query(block):
    with open(os.path.join("sql", "query", "extract_info.sql")) as q:
        sql = q.read()
    conn = None
    try:
        params = config()
        conn = psycopg2.connect(**params)
        cur = conn.cursor()
        cur.execute(sql.format(block))
        rows = cur.fetchall()
        cur.close()
    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
    finally:
        if conn is not None:
            conn.close()
```

```
    return rows

import os
import sys
import database
import logging
from web3.auto import w3
from web3.exceptions import UnhandledRequest
from tqdm import trange
from blockchain import blockexplorer, exceptions

# Read api-key
if os.path.exists("api_code.txt"):
    with open("api_code.txt") as f:
        api_code = f.read().strip()
else:
    api_code = None

# Get logger.
logger = logging.getLogger(__name__)
logger.setLevel(logging.INFO)
fh = logging.FileHandler("build.log")
format = '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
formatter = logging.Formatter(format)
fh.setFormatter(formatter)
logger.addHandler(fh)

if __name__ == "__main__":
    # General Setup at first start.
    if not os.path.exists("progress.txt"):
        logger.info("Start ETL.")
        broken_file = open("broken.txt", "w")
        broken_file.close()
        progress_file = open("progress.txt", "w")
        progress_file.close()
        # First block in chain.
        start_block_nr = 2000000
    # Resume from broken pipe.
    else:
        if os.stat("progress.txt").st_size == 0:
            start_block_nr = 2000000
        else:
            with open("progress.txt", "r") as f:
                # The block to continue from, is the last line in the file.
                for line in f:
                    pass

            start_block_nr = int(line)
            logger.info("Resume ETL at {}".format(start_block_nr))

    # Get latest block and associated number.
    end_block = w3.eth.getBlock("latest")
    end_block_nr = end_block["number"]

    # Fix: sometimes the latest block is unconfirmed and it's number is 0
```

```
    if end_block_nr == 0:
        end_block_nr = start_block_nr + 1000000

    block_keys = ["hash", "gasUsed", "gasLimit", "number", "timestamp"]
    tx_keys = ["hash", "blockHash", "from", "to", "value", "gas", "gasPrice"]
    ip_keys = ["hash", "relayed_by"]

    for current_block_nr in trange(start_block_nr, end_block_nr+1):
        try:
            block = w3.eth.getBlock(current_block_nr)
            database.update(block, 'blocks', block_keys)
            database.update_list(block, 'transactions', tx_keys)
            for tx in database.tx_iterator(block):
                try:
                    transaction = blockexplorer.get_tx(tx, api_code=api_code)
                    database.update(transaction, 'ip_info', ip_keys)
                except exceptions.APIException:
                    continue
                except Exception as e:
                    logger.error(e)
                    continue
            # Save progress
            with open("progress.txt", "a") as f:
                f.write(str(current_block_nr) + "\n")
        except UnhandledRequest as error:
            logger.error(error)
            sys.exit(1)

        except (EOFError, KeyboardInterrupt):
            logger.info("Aborting ETL")
            sys.exit(0)
        except Exception as error:
            logger.error(error)
            with open("broken.txt", "a") as f:
                f.write(str(current_block_nr) + "\n")
            continue

    logger.info("End of ETL.")


import os
import logging
from tqdm import trange
import database
import pandas as pd


# Get logger.
logger = logging.getLogger(__name__)
logger.setLevel(logging.INFO)
fh = logging.FileHandler("query.log")
format = '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
formatter = logging.Formatter(format)
fh.setFormatter(formatter)
logger.addHandler(fh)
```

```
datafile = os.path.join("..", "..", "data", "ethereum-data.csv")
columns = ["block_hash", "block_gas", "gas_limit", "inception_time", "tx_hash",
           "sender", "receiver", "value", "gas_used", "gas_price"]

for block in trange(2000000, 2014806):
    try:
        blockinfo = database.query(block)
        df = pd.DataFrame(blockinfo, columns=columns)
        if os.path.exists(datafile):
            with open(datafile, "a") as f:
                df.to_csv(f, header=False, index=False)
        else:
            df.to_csv(datafile, index=False)
    except Exception as e:
        logger.error(e)

import os
import psycopg2
import pandas as pd

from config import config


datafile = os.path.join("..", "..", "data", "ethereum-data-large.csv")
columns = ["block_hash", "block_gas", "gas_limit", "inception_time", "tx_hash",
           "sender", "receiver", "value", "gas_used", "gas_price"]

with open(os.path.join("sql", "query", "extract_batch.sql")) as q:
    sql = q.read()
conn = None
try:
    params = config()
    conn = psycopg2.connect(**params)
    cur = conn.cursor()
    cur.execute(sql)
    rows = cur.fetchall()
    cur.close()
except (Exception, psycopg2.DatabaseError) as error:
    print(error)
finally:
    if conn is not None:
        conn.close()


df = pd.DataFrame(rows, columns=columns)
df.to_csv(datafile, index=False)

SELECT
  b.block_hash,
  b.block_gas,
  b.gas_limit,
  b.inception_time,
  t.transaction_hash,
  t.sender,
  t.receiver,
  t.value,
```

```
  t.gas_used,
  t.gas_price
FROM (
  SELECT
    block_hash,
    gas_used AS block_gas,
    gas_limit,
    inception_time
  FROM blocks) b
  LEFT JOIN (
  SELECT
    transaction_hash,
    block_hash,
    sender,
    receiver,
    value,
    gas_used,
    gas_price
  FROM transactions) t
  ON t.block_hash = b.block_hash

WITH blocktable AS (
  SELECT
    block_hash,
    gas_used AS block_gas,
    gas_limit,
    inception_time
  FROM blocks
  WHERE block_id = {}
),

transactiontable AS (
  SELECT
    transaction_hash,
    block_hash,
    sender,
    receiver,
    value,
    gas_used,
    gas_price
  FROM transactions
)

SELECT *
FROM blocktable bt
LEFT JOIN transactiontable tt USING(block_hash);


import os
import powerlaw
import networkx as nx
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from matplotlib.ticker import FormatStrFormatter
```

```
data_path = os.path.join(".", "data", "ethereum-data.zip")
general_data_path = os.path.join(".", "data", "export-EtherPrice.csv")
outdir = os.path.join(".", "analysis")

# Load custom dataset
data = pd.read_csv(data_path)
data.inception_time = pd.to_datetime(data.inception_time,
                                     unit="s")

data["day"] = data.inception_time.dt.date
# Convert to ether
data["value"] = data["value"] / 10**18

# Load general dataset
general_data = pd.read_csv(general_data_path,
                           names=["date", "timestamp", "val"], skiprows=[0])
general_data.date = pd.to_datetime(general_data.date)
general_data["day"] = general_data.date.dt.date

# Merge dataframes
data = pd.merge(data, general_data, how="left", on="day")
data.columns
data.agg({"block_hash": "nunique", "tx_hash": "nunique"})

senders = set(data.sender)
receivers = set(data.receiver)

tx_dist = data.sender.value_counts().value_counts(normalize=True)[:10]
breaks = tx_dist.iloc[[0, 2, 3, 9]]

fig, ax = plt.subplots(1)
tx_dist.plot.bar(color="red", ax=ax, rot=0)
ax.set_ylabel("Percentage of wallets")
ax.set_xlabel("Total transactions")
ax.set_yticks(breaks)
ax.yaxis.set_major_formatter(FormatStrFormatter('%.2f'))
ax.hlines(breaks, xmin=-10, xmax=10, color="white", linewidth=1, alpha=1)
ax.set_xlim([-0.5, 9.5])
plt.tight_layout()
fig.savefig(os.path.join("pics", "distribution.png"))
plt.clf()
# Descriptive statistics for custom dataset
desc_cols = ["sender", "receiver", "value", "gas_used", "gas_price"]
desc = data[desc_cols].describe().round(2).iloc[1:, :]
desc.columns = ["value", "gas used", "gas price"]

# Round data and add num users/receivers/senders
desc.to_latex(os.path.join(outdir, "descriptive.tex"))
desc.to_csv(os.path.join(outdir, "descriptive.csv"), index=False)

daily = data.groupby("day").aggregate({"sender": "nunique",
                                       "receiver": "nunique",
                                       "value": [np.mean, np.median, np.std],
                                       "val": "mean"})

receivers = data.loc[:, ["receiver", "day"]]
```

```
receivers.columns = ["user", "day"]
senders = data.loc[:, ["sender", "day"]]
senders.columns = ["user", "day"]

users = pd.concat([receivers, senders])
daily["users"] = users.groupby("day").user.nunique()
daily = daily[["sender", "receiver", "users", "value", "val"]]

# ========================================================================
# Draw the graph
sender = set(data.sender.values)
receiver = set(data.receiver.values)
interactors = sender.intersection(receiver)

loc = data.sender.isin(interactors) & data.receiver.isin(interactors)
interactor_data = data.loc[loc, ["sender", "receiver", "value"]]
interactor_data.shape
len(interactors)

fig, ax = plt.subplots(1, figsize=(15, 15))
sample = interactor_data.sample(2000)
G = nx.from_pandas_edgelist(df=sample,
                            source="sender",
                            target="receiver",
                            edge_attr="value")
G.add_nodes_from(nodes_for_adding=sample.sender.tolist())
nx.draw(G, with_labels=False, ax=ax)
fig.savefig(os.path.join(outdir, "graph-subsample-1000.png"))

fig, ax = plt.subplots(1, figsize=(20, 20))
G = nx.from_pandas_edgelist(df=interactor_data,
                            source="sender",
                            target="receiver",
                            edge_attr="value")
G.add_nodes_from(nodes_for_adding=interactor_data.sender.tolist())
nx.draw(G, with_labels=False, ax=ax)
fig.savefig(os.path.join(outdir, "graph-full.png"))

# Calculate Network metrics
idx = pd.Index(interactors)
outgoing = interactor_data.sender.value_counts().reindex(idx, fill_value=0)
ingoing = interactor_data.receiver.value_counts().reindex(idx, fill_value=0)

edges = pd.concat([outgoing.rename("outgoing"), ingoing.rename("ingoing")],
                  axis=1, levels=["ingoing", "outgoing"])
edges["total"] = outgoing + ingoing

# Centrality
triangles = nx.triangles(G)
edges["triangles"] = pd.Series(triangles).reindex(idx, fill_value=0)
edges["C_in"] = 2*edges["triangles"] / (edges["ingoing"]*(edges["ingoing"]-1))
edges["C_out"] = 2*edges["triangles"]/(edges["outgoing"]*(edges["outgoing"]-1))
edges["C_total"] = 2*edges["triangles"] / (edges["total"]*(edges["total"]-1))

edges.replace([np.inf, np.nan], 0, inplace=True)
avg_cluster = edges[["C_in", "C_out", "C_total"]].mean()
```

```
edges.C_in.plot(kind="hist")
edges.C_out.plot(kind="density", ylim=(0, 1))
edges.C_total.plot(kind="density")
# Fit power law

fig, ax = plt.subplots(1)


def plot_dist(data, ax):
    """Compare theoretical and empirical powerlaw distribution."""
    data = data[data > 0]
    fit = powerlaw.Fit(data, discrete=True)
    fit.plot_pdf(color='b', linewidth=2, ax=ax, label="PDF")
    fit.power_law.plot_pdf(color='b', linestyle="--", ax=ax,
                           label="Power Law fit PDF")

    fit.plot_ccdf(color='r', linewidth=2, ax=ax, label="CCDF")
    fit.power_law.plot_ccdf(color='r', linestyle="--", ax=ax,
                            label="Power Law fit CCDF")

    ax.set_ylabel(r"p")
    ax.legend(loc="lower left")


fig, axes = plt.subplots(ncols=3, figsize=(20, 5))
titles = ["Transactions sent", "Transactions received", "Total transactions"]
datalist = [edges.outgoing, edges.ingoing, edges.total]
for title, ax, data in zip(titles, axes, datalist):
    plot_dist(data.values, ax)
    ax.set_title(title)

fig.savefig(os.path.join(outdir, "power-law-fit.png"))

# Playground Area
# Degree centrality
deg_cent = nx.degree_centrality(G)
edges["deg_centrality"] = pd.Series(deg_cent).reindex(idx)

# Between centrality
bet_cent = nx.betweenness_centrality(G)
edges["betw_centrality"] = pd.Series(bet_cent).reindex(idx)
avg_cluster
avg_cluster.to_latex()

deg_cent = nx.degree_centrality(G)
bet_deg_cent = nx.betweenness_centrality(G)
closeness_cent = nx.closeness_centrality(G)
node_clustering = nx.clustering(G)

avg_clustering = nx.average_clustering(G)


centrality = pd.DataFrame({
    "degree centrality": deg_cent,
    "betweenness centrality": bet_deg_cent,
```

```
    "closeness centrality": closeness_cent,
    "local clustering": node_clustering
})

"""
Create figures for Centrality Measures.
"""

import os
import pandas as pd
from matplotlib import pyplot as plt

data_path = os.path.join("analysis", "centrality_100000.csv")

data = pd.read_csv(data_path, index_col=[0])

data.shape
data.columns = data.columns.str.capitalize()
desc = data.describe().iloc[1:, :].round(4)
desc.to_latex(os.path.join("analysis", "centrality.tex"))

fig, axes = plt.subplots(2, 2)
for col, ax in zip(data.columns, axes.flatten()):
    data[col].plot.box(ax=ax)

fig.savefig(os.path.join("analysis", "centrality.png"))

import os
import networkx as nx
import pandas as pd
from sys import argv

_, sample_size = argv
data_path = os.path.join(".", "data", "ethereum-data.zip")
general_data_path = os.path.join(".", "data", "export-EtherPrice.csv")
outdir = os.path.join(".", "analysis")

data = pd.read_csv(data_path)
data.inception_time = pd.to_datetime(data.inception_time,
                                     unit="s")

data["day"] = data.inception_time.dt.date
# Convert to ether
data["value"] = data["value"] / 10**18

# Load general dataset
general_data = pd.read_csv(general_data_path,
                           names=["date", "timestamp", "val"], skiprows=[0])
general_data.date = pd.to_datetime(general_data.date)
general_data["day"] = general_data.date.dt.date

# Merge dataframes
data = pd.merge(data, general_data, how="left", on="day")
receivers = data.loc[:, ["receiver", "day"]]
receivers.columns = ["user", "day"]
senders = data.loc[:, ["sender", "day"]]
```

```
senders.columns = ["user", "day"]

# ================================================================================
# Draw the graph
sender = set(data.sender.values)
receiver = set(data.receiver.values)
interactors = sender.intersection(receiver)

loc = data.sender.isin(interactors) & data.receiver.isin(interactors)
interactor_data = data.loc[loc, ["sender", "receiver", "value"]]

print("Initialize graph")
sample = interactor_data.sample(10000)
G = nx.from_pandas_edgelist(df=sample,
                            source="sender",
                            target="receiver",
                            edge_attr="value")
G.add_nodes_from(nodes_for_adding=sample.sender.tolist())

print("Compute Centrality-Measures")
deg_cent = nx.degree_centrality(G)
bet_deg_cent = nx.betweenness_centrality(G)
closeness_cent = nx.closeness_centrality(G)
node_clustering = nx.clustering(G)

avg_clustering = nx.average_clustering(G)
print("Average clustering coefficient:", avg_clustering)

centrality = pd.DataFrame({
    "degree centrality": deg_cent,
    "betweenness centrality": bet_deg_cent,
    "closeness centrality": closeness_cent,
    "local clustering": node_clustering
})

print("Save to .csv")
centrality.to_csv(os.path.join(outdir, "centrality_"+str(sample_size)+".csv"))

if(!require("data.table")) install.packages("data.table"); library("data.table")
if(!require("dplyr")) install.packages("dplyr"); library("dplyr")
if(!require("ggplot2")) install.packages("ggplot2"); library("ggplot2")
if(!require("lubridate")) install.packages("lubridate"); library("lubridate")
if(!require("maptools")) install.packages("maptools"); library("maptools")
if(!require("scales")) install.packages("scales"); library("scales")
library(xtable)
# if(!require("gridExtra")) install.packages("gridExtra"); install.packages("gridExtra")
# if(!require("ggthemes")) install.packages("ggthemes"); install.packages("ggthemes")

# Data transformations
e_w_rate <- 1000000000000000000 #1 Ether equals x wei
s_w_rate <- 1000000000000 #1 Szabo equals x wei

# # dt <- read.csv("data/ethereum_data.csv", colClasses = "character")
dt <- read.csv("ethereum-data-large.csv", colClasses = "character")
s1 <- summary(dt)
# # colnames(dt)[colnames(dt)=="block_bas"] <- "block_gas"
```

```
dt[c("block_hash", "tx_hash", "sender", "receiver")] <- lapply(dt[c("block_hash", "tx_hash", "
dt[c("block_gas", "gas_limit", "inception_time", "value", "gas_used", "gas_price")] <- lapply(
# # dt[c("block_gas", "gas_limit", "value", "gas_price")] <- dt[,c("block_gas", "gas_limit", "
dt$inception_time <- as.POSIXct(dt$inception_time, origin="1970-01-01")
dt$time <- trunc(dt$inception_time, units = "hours")
dt$time <- as.POSIXct(dt$time, origin="1970-01-01")
dt$day <- trunc(dt$inception_time, units = "days")
dt$day <- as.POSIXct(dt$day, origin="1970-01-01")
s2 <- summary(dt)

dt <- dt[!is.na(dt$value),]
# # dt[is.na(dt)] <- 0 # BE CAREFULL WITH THIS
dt$fee <- dt$gas_used*dt$gas_price


# descriptives <- data.frame()
# descriptives$mean <- median(dt[,c("value","gas_used")], na.rm = TRUE)
# median(dt$value)
dt_sum <- lapply(dt[,c("value","gas_used","gas_price", "fee")],sum)
dt_median <- lapply(dt[,c("value","gas_used","gas_price", "fee")],median)
dt_max <- lapply(dt[,c("value","gas_used","gas_price", "fee")],max)
#
# dt <- data.table(dt)

# Data description ----
s3 <- summary(dt)

# Graphs

hist(dt$gas_used, col = "orange", border = "grey",
     main = "Distribution of Gas used per Transaction",
     xlab = "Amount of Gas per Transaction, in wei", ylab = "Number of Transactions", ylim = c
boxplot(dt$gas_used,
        # main = "Boxplot of Gas used per Transaction",
        xlab = "Amount of Gas per Transaction, in wei")

# hist(dt$gas_price)
# boxplot(dt$gas_price)

hist(dt$fee/e_w_rate, col = "orange", border = "grey",
     # main = "Distribution of Fees paid per Transaction",
     xlab = "Fee per Transaction, in ether", ylab = "Number of Transactions", ylim = c(0,700),
boxplot(dt$fee/e_w_rate,
        # main = "Boxplot of Fees paid per Transaction",
        xlab = "Fee per Transaction, in ether")

# hist(dt$block_gas, col = "blue", main = "Distribution of Gas Used within a Block", xlab = "A
# boxplot(dt$block_gas, main = "Boxplot of Gas Used within a Block", xlab = "Amount of Gas Use
# # CAREFULL!! THIS TAKES TX to make a calculation

hist(dt$value/e_w_rate, col = "orange", border = "grey",
     # main = "Distribution of Value sent per Transaction",
     xlab = "Value of a Transaction, in ether", ylab = "Number of Transactions", ylim = c(0,50
boxplot(dt$value/e_w_rate,
        # main = "Boxplot of Value sent per Transaction",
        xlab = "Value of a Transaction, in ether")
```

```
# Add fee calculation

tx_n <- nrow(dt)
tx_volume <- sum(dt$value, na.rm = TRUE)
tx_average_value <- tx_n/tx_volume
tx_n; tx_volume; tx_average_value

# hist(dt$value/1000000000000000000,xlab = "Value of a TX", main="Distribution of TXs values",
# summary(dt[,c("value","gas_used")])


# dt_info <- matrix(nrow = 2, ncol=3)
# colnames(dt_info) <- c('period','tx_n', 'tx_volume')
# dt_info <- as.table(dt_info)


# blocks_n <- length(unique(unlist(dt$block_hash)))
# average_tx_per_block <- tx_n/blocks_n
# average_tx_volume_per_block <- tx_volume/blocks_n


block_descriptives <- dt %>%
  group_by(block_hash) %>%
  summarize(block_tx=n(), block_tx_perc=n()/tx_n*100, block_volume=sum(value),
            block_volume_perc=sum(value)/tx_volume*100, block_average_tx_value=mean(value)) %>%
  arrange(desc(block_tx))
summary(block_descriptives)

dt_sum$block_tx <- sum(block_descriptives$block_tx)
dt_sum$block_volume <- sum(block_descriptives$block_volume)

dt_median$block_tx <- median(block_descriptives$block_tx)
dt_median$block_volume <- median(block_descriptives$block_volume)

dt_max$block_tx <- max(block_descriptives$block_tx)
dt_max$block_volume <- max(block_descriptives$block_volume)

hist(block_descriptives$block_tx, col = "orange", border = "grey",
     # main = "Distribution of Transactions' Number per Block",
     xlab = "Number of Transactions", ylab = "Number of Blocks", ylim = c(0,7000))
hist(block_descriptives$block_volume/e_w_rate, col = "orange", border = "grey",
     # main = "Distribution of Transactions' Volume per Block",
     xlab = "Volume of Transactions, in ether", ylab = "Number of Blocks", ylim = c(0,100), br
# boxplot(block_descriptives)


time_descriptives <- dt %>%
  group_by(day) %>%
  summarize(tx_number=n(), tx_perc=n()/tx_n*100, tx_vol=sum(value, na.rm = TRUE),
            tx_vol_perc=sum(value, na.rm = TRUE)/tx_volume*100, tx_average_value=mean(value, n
  arrange(desc(day))
time_descriptives <- time_descriptives[-c(1, nrow(time_descriptives)), ] # remove first and la
summary(time_descriptives)


dt_sum$tx_number <- sum(time_descriptives$tx_number)
```

```
dt_sum$tx_volume <- sum(time_descriptives$tx_vol)

dt_median$tx_number <- median(time_descriptives$tx_number)
dt_median$tx_volume <- median(time_descriptives$tx_vol)

dt_max$tx_number <- max(time_descriptives$tx_number)
dt_max$tx_volume <- max(time_descriptives$tx_vol)

# ggplot(time_descriptives,aes(day,tx_number,group=1))+ theme_bw() + geom_point()+
#   scale_x_datetime(labels = date_format("%y:%m:%d")) +
#   theme(axis.text.x = element_text(angle=90,hjust=1))

ggplot(data=time_descriptives,aes(x=day, y=tx_number)) +
  geom_line(colour="blue") +
  # ggtitle("Number of Transactions over Time") +
  ylab("Number of Transactions") +
  xlab("Date")

ggplot(data=time_descriptives,aes(x=day, y=tx_vol/e_w_rate)) +
  geom_line(colour="blue") +
  # ggtitle("Volume of Transactions over Time, in ether") +
  ylab("Volume of Transactions") +
  xlab("Date")


# Major wallets
major_senders <- dt %>%
  group_by(sender) %>%
  summarize(sender_tx=n(), sender_tx_perc=n()/tx_n*100, sender_volume=-sum(value),
            sender_volume_perc=-sum(value)/tx_volume*100, sender_average_value=-mean(value)) %
  arrange(desc(sender_tx))
colnames(major_senders)[1] <- "wallet"
summary(major_senders)

major_receivers <- dt %>%
  group_by(receiver) %>%
  summarize(receiver_tx=n(), receiver_tx_perc=n()/tx_n*100, receiver_volume=sum(value),
            receiver_volume_perc=sum(value)/tx_volume*100, receiver_average_value=mean(value))
  arrange(desc(receiver_tx))
colnames(major_receivers)[1] <- "wallet"
summary(major_receivers)

major_senders <- data.table(major_senders)
major_receivers <- data.table(major_receivers)
setkey(major_receivers,wallet)
setkey(major_senders,wallet)
m_w <- merge(major_senders,major_receivers, all = TRUE)
m_w[is.na(m_w)] <- 0
m_w$number_tx <- m_w$sender_tx + m_w$receiver_tx
m_w$vol_tx <- -m_w$sender_volume + m_w$receiver_volume
m_w$balance <- m_w$sender_volume + m_w$receiver_volume
summary(m_w)


top50_sender_tx <- head(m_w[order(-sender_tx),c("wallet","sender_tx")], 50)
top50_sender_tx$cum <- cumsum(top50_sender_tx$sender_tx)
```

```
top50_sender_tx$perc <- top50_sender_tx$cum/tx_n*100
plot(top50_sender_tx$perc, ylim = c(0,100),
     # main = "Cumulative % of TXs' Number covered by Major Senders",
     xlab = "Top 50 Senders", ylab = "Percentage of all Transactions Number")

top50_receiver_tx <- head(m_w[order(-receiver_tx),c("wallet","receiver_tx")], 50)
top50_receiver_tx$cum <- cumsum(top50_receiver_tx$receiver_tx)
top50_receiver_tx$perc <- top50_receiver_tx$cum/tx_n*100
plot(top50_receiver_tx$perc, ylim = c(0,100),
     # main = "Cumulative % of TXs' Number covered by Major Receivers",
     xlab = "Top 50 Receivers", ylab = "Percentage of all Transactions Number")


top50_sender_volume <- head(m_w[order(sender_volume),c("wallet","sender_volume")], 50)
top50_sender_volume$cum <- -cumsum(top50_sender_volume$sender_volume)
top50_sender_volume$perc <- top50_sender_volume$cum/tx_volume*100
plot(top50_sender_volume$perc, ylim = c(0,100),
     # main = "Cumulative % of TXs' Volume covered by Major Senders",
     xlab = "Top 50 Senders", ylab = "Percentage of all Transactions Volume")

top50_receiver_volume <- head(m_w[order(-receiver_volume),c("wallet","receiver_volume")], 50)
top50_receiver_volume$cum <- cumsum(top50_receiver_volume$receiver_volume)
top50_receiver_volume$perc <- top50_receiver_volume$cum/tx_volume*100
plot(top50_receiver_volume$perc, ylim = c(0,100),
     # main = "Cumulative % of TXs' Volume covered by Major Receivers",
     xlab = "Top 50 Receivers", ylab = "Percentage of all Transactions Volume")


plot(top50_sender_tx$perc, ylim = c(0,100), col = "blue",
     # main = "Cumulative % of TXs covered by Major Users",
     xlab = "Top 50 Senders (blue) and Top 50 Receivers (orange)", ylab = "Percentage of covere
points(top50_receiver_tx$perc, ylim = c(0,100), col = "orange")

plot(top50_sender_volume$perc, ylim = c(0,100), col = "blue",
     # main = "Cumulative % of Volume covered by Major Users",
     xlab = "Top 50 Senders (blue) and Top 50 Receivers (orange)", ylab = "Percentage of covere
points(top50_receiver_volume$perc, ylim = c(0,100), col = "orange")

plot(top50_sender_tx$perc, ylim = c(0,100), col = "blue",
     # main = "Cumulative % of TXs and Volume covered by Major Senders",
     # xlab = "Top 50 Senders in terms of Transactions (blue) and Volume (orange)",
     ylab = "Cumulative Percentage")
points(top50_sender_volume$perc, ylim = c(0,100), col = "orange")

plot(top50_receiver_tx$perc, ylim = c(0,100), col = "blue",
     # main = "Cumulative % of TXs and Volume covered by Major Receivers",
     # xlab = "Top 50 Receivers in terms of Transactions (blue) and Volume (orange)",
     ylab = "Cumulative Percentage")
points(top50_receiver_volume$perc, ylim = c(0,100), col = "orange")


intersect_volume <- intersect(top50_receiver_volume$wallet,top50_sender_volume$wallet)
intersect_tx <- intersect(top50_receiver_tx$wallet,top50_sender_tx$wallet)

intersect_senders <- intersect(top50_sender_tx$wallet,top50_sender_volume$wallet)
intersect_receivers <- intersect(top50_receiver_tx$wallet,top50_receiver_volume$wallet)
```

```
top50_wallet_tx <- head(m_w[order(-number_tx),c("wallet","number_tx")], 50)
top50_wallet_tx$cum <- cumsum(top50_wallet_tx$number_tx)
top50_wallet_tx$perc <- top50_wallet_tx$cum/tx_n*100/2
plot(top50_wallet_tx$perc, ylim = c(0,100),
     # main = "Cumulative % of TXs' Number covered by Major Wallets",
     xlab = "Top 50 Wallets", ylab = "Percentage of all Transactions Number")

# top100_wallet_volume <- head(m_w[order(-vol_tx),c("wallet","vol_tx")], 100)
# top100_wallet_volume$cum <- cumsum(top100_wallet_volume$vol_tx)
# top100_wallet_volume$perc <- top100_wallet_volume$cum/tx_volume*100/2
# plot(top100_wallet_volume$perc, main = "Cumulative % of TXs' Volume covered by Major Receive

hist(m_w$balance/e_w_rate, col = "orange", border = "grey", ylim = range(0,10),
     # main = "Distribution of Balances within Studied Period",
     xlab = "Net Balances of Wallets, in ether", ylab = "Number of Wallets",breaks = 50)



# a <- unlist(broad_data$wallet)
# nrow(broad_data) == length(a)
#
#
# # List of Nodes and edges
# nodes <- data.table(a)
# colnames(nodes) <- "Nodes"
#
# edges <- data.table(dt[,c("sender","receiver")])
# edges <- edges[edges$sender != "",]
# colnames(edges) <- c("Source", "Target")
#
# write.csv(nodes,"data/nodes.csv")
# write.csv(edges,"data/edges.csv")
#
# # Furthe inspiration
# order_weekday_abs <- train_broad_data[order ==1, NROW(lineID), by = list(weekday)]
```