POLITECNICO DI TORINO

Master Degree in Computer Engineering

Master Degree Thesis

Cryptocurrency and Blockchain: An experimental analysis of the Miners' Ecosystem

Candidate:
Manuel Rafeli

Advisors:
Prof. Squillero Giovanni
PhD. Edoardo Fadda

Academic Year 2017/2018

Acknowledgement

In primis vorrei ringraziare il prof. Giovanni Squillero per avermi dato la possibilità di realizzazione una tesi su questo argomento.

Ringraziamento speciale al PhD. Edoardo Fadda per la sua immensa disponibilità e soprattutto per esser stato un riferimento importante in tutte le fasi dello sviluppo di questo lavoro di tesi nonostante la distanza logistica.

Alla azienda in cui lavoro, IFM srl di Catanzaro, che mi ha concesso la possibilità di poter dedicare del tempo allo studio, e a tutti i miei colleghi che mi hanno supportato in questa sfida.

Voglio poi dire Grazie a Mamma e Papà per aver sempre creduto in me, per avermi costantemente spronato a dare il meglio e per aver sempre sostenuto le mie iniziative.

A mio fratello e mia sorella, che con il suo supporto tecnico ed emotivo ha reso meno tortuoso il cammino verso questo traguardo.

Infine un grazie va ai miei amici, tutti, per aver creduto in me e elogiatomi facendo sentire ancora più importante quanto fatto.

Abstract

In 2008, a man under the alias of Satoshi Nakamoto invented the Bitcoin, an electronic cash system completely decentralized that does not rely on a central authority for currency issuance or settlement and validation of transactions. The technology which avoids this system is called 'Blockchain', an incorruptible digital distributed ledger that provides a secure way of making and recording transactions. In this thesis our goal is to study the blockchain technology and its main aspects focusing on miner's features in such a way that we are able to build a blockchain simulator which permits to enforce various miner behaviour patterns and analyzing the outputs to understand better miners' ecosystem.

Contents

Chapter 1

Introduction

1.1 Blockchain

Blockchain technology is a set of concepts and techniques to implement a system that provides a secure way of making and recording transactions, agreements and contracts. It is based on a distributed peer-to-peer network and its name derives from the way of storing data in a digital ledger. It is also called Distributed Ledger Technology and it is below of Bitcoin and all cryptocurrencies.

Blockchain stores information as transactions into a global digital ledger which is distributed and replicated in each node of the same network. The data lake consists of a chain of blocks where each of them contains some data and when a change happens it is distributed globally because anyone can participate in verifying the transaction. The blocks are concatenated each other avoiding the possibility to remove or to alter existing data.

By Cryptography, the recordings are secure and essentially unalterable on the ledger. It is important to verify the authorship of transaction using a form of digital signature.

According to be public, the blockchain is accessible to anyone and it is completely transparent, all of participants can read the blocks and monitoring the data exchanges between the users. In this case, blockchain is called *Permissionless*. On the other hand, a *Permissioned* blockchain is a private network where each participants could be a different visibility level.

1.2 Permissionless vs Permissioned

Permissionless blockchains are systems where anyone can participate without receive any permission. They are open, not created to be controlled by a system admin and they have not a owner [1]. Any person can be-

come a participant downloading the code and running a public node on own local device which permits to start validating transactions in the network, participating in the consensus process, or sending transactions through the network. Running the client on local make users actors of the network as a simple user, if they send or receive valid transaction, or as a Miner, if the participant complete the consensus process. In public blockchains, it does not need to maintain central servers so there are not infrastructure costs. Typical examples are cryptocurriences such as Bitcoin, Ethereum or IOTA.

By contrast, Permissioned blockchain has evolved as an alternative way to permit anyone to take part of the ecosystem. Participating in the process of verifying transactions is allowed only for particular group of entities, defined as Trusted, selected by network admin who share a mutual goal. They should be identified and their roles need to be defined if they are to play a part in the network. A Permissioned blockchain can use the traditional Byzantine-fault tolerant consensus which is a protocol based on Byzantine General's Problem. Permissioned ledgers permit to define special rules about data visibility ensuring more transaction privacy. It means that these blockchains can be controlled introducing the concept of Governance wanted by institutions and big companies. These features reduce transaction costs and data redundancies causing greater speed of the network. The best example of Permissioned blockchain is Hyperledger. [2]

1.3 Distributed Ledger Technologies Frameworks

Blockchain landscape has evolved since Bitcoin, first blockchain implementation, was introduced by Satoshi Nakamoto. At first, it was dominated by different blockchain protocols payments oriented but soon new various

protocols were born keeping in mind the different use cases and objectives introducing important technological innovations. Nowadays, new companies are supporting distributed ledger technologies and they are implementing enterprise solutions for business uses such as supply chain or financial markets.

Bitcoin is the earliest blockchain protocol and it is blockchain first oriented because you work directly with the given blockchain tools and stack. Bitcoin is a Permissionless blockchain and uses a Proof-of-Woork consensus algorithm. Immutable data is ensured by using cryptographic hash function, digital signature and private-and-public key encryption. [3]

Ethereum philosophy is similar to Bitcoin, it is the first innovation blockchain from Bitcoin. It uses also a Permissionless blockchain and PoW consensus algorithm but, unlike Bitcoin, which was built for allowing cryptopayment transactions over a decentralized network, Ethereum blockchain was designed to permit developers to create their own blockchain projects, including their own cryptocurrencies. It is based on Smart contracts, pieces of code which allows execution of legal functions according required conditions.

IOTA is a next-generation blockchain focused on provide trust transactions between the machines within Internet of things context. Its particular feature is that there are not miner but all participant of the network participate actively in the consensus mechanism. In fact, a node who wants to make a transaction has to approve two past transactions. That makes IOTA more decentralized than any blockchain.

This framework improves scalability of the system achieving high transaction throughput exploiting parallel validation of transaction with no limit as to the number of transaction that can be confirmed in a certain interval. [4]

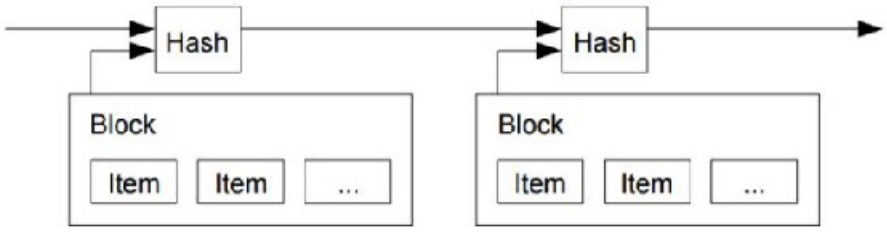
As its website say, Hyperledger is an open source collaborative effort created to advance cross-industry blockchain technologies. It is a global collaboration, hosted by The Linux Foundation, including leaders in finance, banking, IoT, supply chain, manufacturing and technology. It implements a Permissioned blockchain and follows the specific requirements of large enterprises such as high scalability, transaction speed and trusted entities should join the network. [5]

1.4 Blocks

Blockchain is a ledger of transactions shared across multiple computers. Technically, it is a huge file divided in a set of data which are collected and processed to fit in a *block* through a process called mining.

A block contains a set of transactions and some other attributes needed to sustain correctly the mechanism. Data are stored in these transactions which are format depending on type of blockchain we are using.

Each block has a digital fingerprint created using a cryptographic hash based on data inside transactions and block's attributes. It will be contained in the next block as an attribute of it so that blocks can form a chain from the first block ever (known as the Genesis Block) to the formed block.

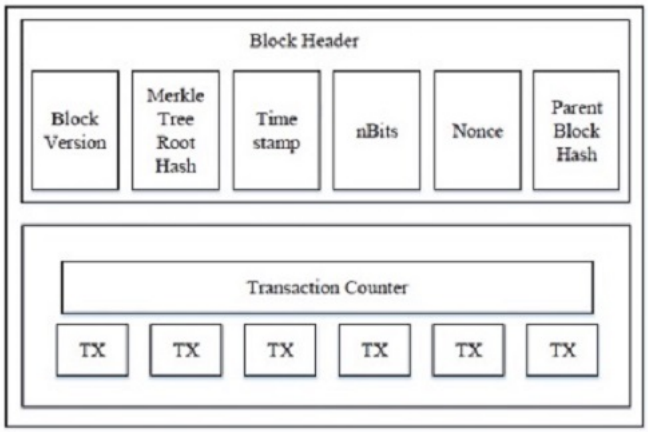


A visual illustration of blockchain.
Source: Bitcoin: A Peer-to-Peer Electronic Cash System, S. Nakamoto

Hash value of the block is unique in the network and it permit to identify the block uniquely on the ledger. It is not contained inside the block's data structure but it is computed by each node when they receive the block from neighbor. Block's hashes might be stored in a separate database to retrieve easily blocks from disk. Another way to identify a block is by height in the blockchain. Height is the position of the block inside the chain, for example first block has 0 height value. When a block is added to the chain, its height will be higher of one than its parent block. Unlike the block hash, the height does not identify uniquely the block because more block could compete for the same position in the chain. The height value does not also store inside the block's data structure but it might retrieve from database.

A general block structure is composed of a header and a body. In the figure we can see that the header is structured by followed metadata :

- Block version: track bitcoin protocol upgrades
- Merkle Tree Root: data structure which summarize all transaction in the block
- Timestamp: timestamp creation of the block
- nBits: difficulty target for this block used in the Proof-of-Work Algorithm
- Nonce: counter used in The Proof-of-Work algorithm counter
- Previous block hash: a reference to the parent block in the blockchain



The body contains a list of transactions which are limited on depend of the size of the block and the size of each

1.5 Trustless

The main feature of blockchain is to be *Trust*. Generally, if two people who don't know each other want to exchange something as money, they entrust to a third part as a bank to guarantee the exchange. It happens because people don't trust each other and it needs the third part to oversee it.

Blockchain eliminates the need for a third party because its property permits to have a distributed trustless consensus which guarantee that transactions is valid and safe for both.

Two important aspects make possible this innovation:

- Cryptography, every transaction is signed by the author using a digital signature so nobody can crate a transaction which doesn't own himself. This form of encryption requires consensus from everyone in the network to unlock and make changes to.

- Distributed, the digital ledger contains all transactions and each node has a copy of it. The new transactions are distributed on the whole network and each node, after validating, aggregate them in a block which will be added in the ledger only after reaching a *distributed consensus* on it. This process ensures that the ledger will be immutable.

1.6 Cryptography

Cryptography is a method of storing and transmitting data in a particular form in such a way that only those for whom it is intended can read and process it. Blockchain uses two technique already known in the modern system: Asymmetric Cryptography and Hashing.

Asymmetric Cryptography permits to identify uniquely the author of transaction guaranteeing the contents of it have not been altered in transit. This type of cryptography, also know as Public-Key Cryptography, has made safer the concept of to use only one secret to protect data. It uses two related keys respectively called public and private key where the first may be freely distributed, while its paired private key had to remain a secret. They are strongly correlated with each other through some mathematical relationship.

In particular, Digital signature implements the previous concepts. If you want to sign a digital document by a digital signature, you have to do a fingerprint of it, an hash typically, and encrypting this fingerprint using the signer's private key. The test of the sign are made decrypting the fingerprint signed using the signer's public key and comparing it with a fingerprint of the original document. If they are not equal the sign is not valid or the digital document has been corrupted.

In the blockchain technology, users own public and private key which are stored in a digital wallet.

Public key, also called *address*, identify the wallet in the network and it "belongs" to the network after first transaction refers to it is written on the blockchain. If you want to exchange any things with another user you have to create the transaction using him address.

3Au6uLAKnRB6id5kaxGqZ5mf8mkrjEVPT6

example of the address

Private key is used by user to sign the transaction created. It is vital to maintain it in a secure place because the private key's owner proves that data refer to a certain public key belong to him. In particular, blockchain exploit the digital signature in this way: when a transaction is created, the author signs digitally the transaction using his private key. This signature ensures that only the owner of the wallet has created the transaction.

As we have seen previous, Hash function is an another cryptographic method used by blockchain technologies. It implements the concept of taking an arbitrary amount of input data, applying an algorithm to it, and generating a fixed-size output data called hash. The hash represents a mark of the input data which will be unique inside a particular context. In the blockchain, hash function permits to identify uniquely a certain block in a chain which will be include in the header of next block. The result is that each block are related each of other because hash algorithm is applied on the block's transactions plus the previous block's hash and the last hash calculated represent the current state of the world.

1.7 Distributed

Blockchain is based on the peer-to-peer distributed network with no central authority figure where each peer or node gives a contribution to maintain the net safe. It is a architectural change because there is no more a central system that preserves all data but we have distributed peers which have the same copy of the data. This architecture limits the risks of errors or changes on data but also attacks on the system.

The main purpose of the distributed network is to achieve a "distributed consensus". In a central system all decisions are taken by the leader or a few superior actor. In the blockchain anyone does not prevail on another because all nodes have the same role and it needs to obtain a global consensus.

Any participants that receive a valid transaction which has not seen before use the technique known as flooding for propagate the data. It consists to forward immediately the transaction to all other participants to which it is connected. In this circumstance, the transactions are not still valid but it needs that a node aggregates some transactions creating a block and reaching the distributed consensus on them.

Consensus permits that an agreement is reached using a decision-making process which can benefit the entire group in the best interest of the whole and it is needed when it wants to create a more equal and fair society. To reach a consensus on a distributed system is a problem knows as "Byzantine Generals' Problem." It consists of trying to agree on a course of action or the state of a system by exchanging information over an unreliable and potentially compromised network.

Consensus mechanism is the method by which Byzantine Generals' Problem is solved. It exists a lot of different consensus algorithm such a Proof-of-Stake or Proof-of-Elapsed-Time but the most used is the Proof-of-Work

which are used in Bitcoin Blockchain.

1.8 Mining

Transactions just created are propagated across the network but they does not become part of the blockchain until they are verified and included in a block by a process called mining. Mining is the process crucial to achieve a consensus on decentralized networks to prove the fairness of elections, lotteries, asset registries, digital notarization, and more. It is done by *miners* and its function is to generate new tokens in a decentralized and distributed system. Every participant of the network could be a Miner and each of them competes to be the first to create a new block. In the most cases, first miner who creates a new block adds it into blockchain and he is remunerated with a fee. This is a blockchain with a reward system because miners are encouraged to mining by coins obtained. Exists a lot of protocol which implement mining process such as Proof of Work or Proof of Stake.

1.8.1 Proof of Work

Bitcoin blockchain uses a Proof-of-Work algorithm for reaching a consensus. Miners have to find a hash value that must be smaller than a certain number which is the difficulty level set by the network. Hash value is calculated with the header of the new block plus a nonce. Nonce is a number which is changed in every iteration of hashing function until it reaches the right number.

The difficulty level is dynamically tuned by the Bitcoin network protocol and its value is crucial to maintain steadily that one block is produced every ten minutes. After 2016 blocks, this value is changed calculating a new one

based on time spent to produce the previous 2016 blocks.

Proof-of-Work in Bitcoin blockchain has a key role because it proves that you consumed power to create a new block introducing a *economy barrier* to protect the network. If an malicious would want to alter the data in one block, all previous of that block must be re-written and a huge amount of calculation is necessary spending a lot of money.

However, the blockchain may use other mechanisms of consensus. Instead of hash functions, for example, a blockchain may use Scrypt for proof-of-work algorithm.

1.8.2 Proof of Stake

Proof of Stake is an alternative protocol for process mining. It resolves the main critical problem of Proof of Work which encourage the global energetic expenditure and favors the centralization of computing power in few places on the world. Unlike the proof-of-Work, where the algorithm uses the power competing to solve the block, with th Proof-of-Stake, who create a new block is chosen depending on coin-age and randomness. Coin-age is the amount of tokens owned by user and it gives a guarantee in creating new block. The larger is the amount of tokens owned by a user, the greater the chances that he is not violating the system and the more likely he is to behave in an optimal way.

This protocol needs three feature:

- A sufficient number of miners nodes within the network
- A minimum amount of tokens associated with each miner's profile
- The need for there to be already in the circle of tokens in the network

In the most cases where Proof-of-stake is used, cryptocurrencies are already mined and users enter on by participation. It means that the total amount of cryptos is established at the beginning. A miner, when create a block, does not get any reward but he takes only the transaction fees.

As we observed, the main problem of this solution is that is needed to have a minimum amount of tokens within the network but low energy consumption is a great advantage.

1.8.3 Mining Pool

Mining Pool is a set of miner which contribute together to the find a new block, and then split the block reward according the contributed processing power.

Generally, in the competition of finding new block the miner works alone but when he is not well equipped, in terms of hardware for PoW or of stake for PoS, this search seems to a lottery. Rising of difficulty network over time makes increasingly creating new block and obtaining the reward. It means that a miner who invests a part of own budget on mining could not return the investment at the end.

Mining Pool is the right compromise to reduce the enormous risk and amortizing the cost of equipment. Participating in a mining pool, it is able to earn a regular payouts over that period. Peers who take part to a mining pool are coordinated by pool server which uses a specialized pool-mining protocols. The pool server synchronizes the effort of miners' equipment to search a solution for the candidate block. Once find new block, it will share the reward among peers who belong to a mining pool based on their mining contribution.

Payment could be done in two ways: sending directly to miners through

his wallet when they ask for them or setting the coinbase transaction to pay the block reward to their wallet once a block is found. The latter has the advantage that it never happen that any Bitcoin are stolen on your pool server.

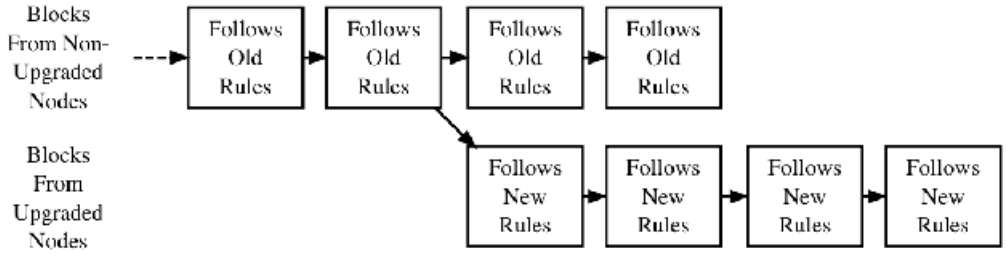
1.9 Fork

In decentralized system can occur that data structure of each node is not always the same. Blockchain follows this rule in cause of its propagation mechanism of new blocks. Different nodes might receive different blocks at the same time causing the peers have different perspectives of the blockchain. It means that each node must maintain at least two different chain but they will consider as main chain the one with the longest chain or greatest cumulative work. When more than one chain exist means that a Fork has happened. Indeed, it consists in a temporary diverge between versions of the blockchain which will be resolved by re-convergence in one single chain after one or more blocks are mined. Fork may happen when as a result of transmission delays in the entire network or when two or more miners mine a block with the same height.

1.9.1 Hard Fork

This type of fork is a naturally fork because occurs as part of the consensus operation. On the other hand, exists another type of fork called Hard fork which does not occur naturally and does not converge onto a single blockchain. It happens when part of the network want to change the consensus rules because they want to fix some bugs or deliberate change in the implementation. In this scenario, the chains of the different version evolve

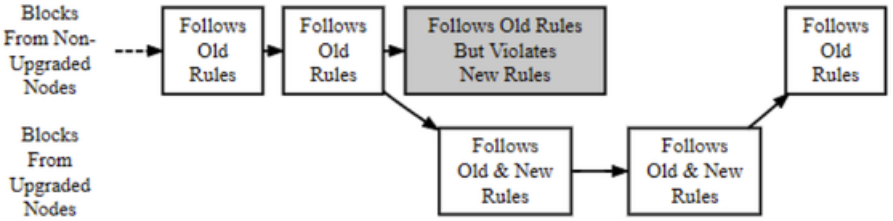
independently. and it is not "forward compatible".



Hard fork requires coordination between all nodes in the system. It means that each participant in the new system must upgrade his client so they can implement new consensus rules and switching on the new blockchain version. Bitcoin Cash and Bitcoin Gold are examples of hard fork because they adopted a different competing implementation introducing new rules.

1.9.2 Soft Fork

When the change is implemented in such a way that an old client still sees the transaction or block created by new client as valid, it is called Soft Fork. Soft fork updates the consensus rules but it is forward-compatible to the previous. It means that older client can continue to operate in consensus with the new rules.



A Soft Fork: Blocks Violating New Rules Are Made Stale By The Upgraded Mining Majority

The fundamental aspect of a soft fork is that it does not require that all clients must upgrade and does not force non-upgraded nodes out of consensus.

In the case of a deliberate change to the consensus rules, a software fork precedes the hard fork. However, for this type of hard fork to occur, a new software implementation of the consensus rules must be developed, adopted, and launched.

1.10 Smart contracts

Smart contracts are an evolution of the initial blockchain and the concepts will be the leader in blockchain technology in the next years. They was introduced by Ethereum and it is the main innovation of its platform.

1.10.1 Definition

Traditional contracts are created by professional legal and they are written in a legal language that often result to be ambiguous. Often people do not understand the legal language and if one party does not respect the terms of the contract, the public judiciary system resolves the issue for them. This process has a cost in terms of resources, time and amount of paperwork used.

On the other hand, smart contracts are completely digital and they are written by programming languages. They are easy to understand because there is not much technical legal language and the code dictate the terms of consequences of the contract. Moreover, none third party is necessary for the enforcement of the contract. [6]

Once smart contract is implemented, it will put on the blockchain causing its activation automatic. After that, it become part of the chain and there is no chance that a third part may modify or stopping it. It will make automated deductions and will perform the consequences of the terms written on it based on the event which will happen.

A debit card is a typical example where we can look the powerful of smart contracts. Whenever the owner uses his debit cart, the third party will decrease his amount on bank account making automated deduction. The same behaviours are done by smart contract but the difference is that occurs in distributed way. Exploiting feature of blockchain, smart contracts can execute without that a third party influences this execution in contrast of the debit card where the bank can decide to act on the account. Moreover, transactions derive from smart contrast are safe because it is impossible to hack a blockchain.

One possible disadvantage of smart contract is that on the public blockchain all data written can be seen by anyone. It may be a problem if the smart contract contains information which must be not available to the public. For example, a company that makes a smart contract with some supplier and it does not want that his concurrent acquire information about that.

1.10.2 Dapp

Dapp, also known as a decentralized application, are the first real uses of smart contracts and they are in the Ethereum marketplace. A dapp is a set of smart contracts which define the function of the decentralized application.

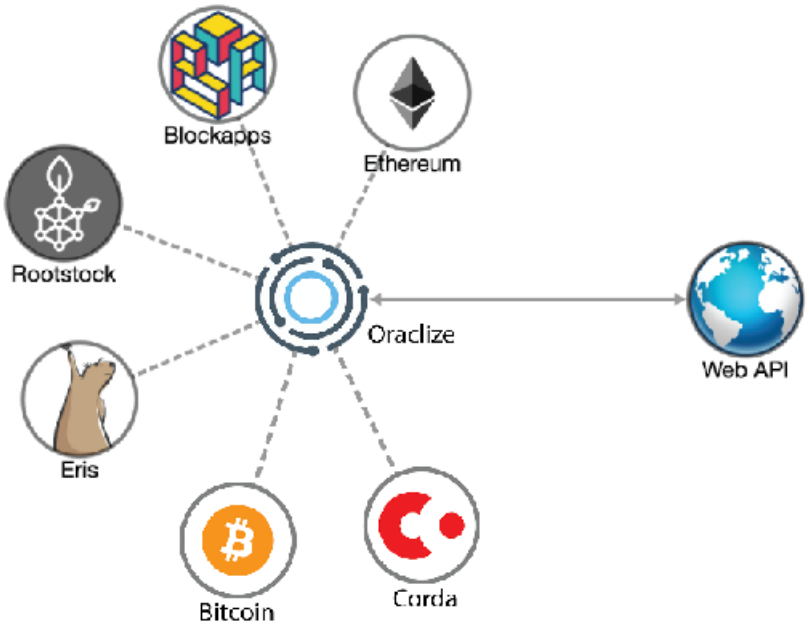
It was introduced by Ethereum platform and they are written in Solidity, a programming language much similar to Javascript. After its introduction, other platform was born for implementing Daap design such as Hyperledger or NEO.

Daap has the three main following features:

- Open-source: its code is totally open-source and anyone can see how it is written.

- Decentralized: it makes available by more servers, not only one, allowing developers to distribute the profit among them
- Privacy: it does not need of personal information because it works on blockchain which identifies an user by an public key (address).

A Dapp is divided in back-end and front-end. Back-end stays on the decentralized peer-to-peer network, the blockchain, while Front-end is a interface that collects data and interacts with the users. The latter must be implemented in such a way to be compatible with back-end. Oraclize, also defined as Web API, is the one of the most famous available front-end and it takes care of fetch data into Ethereum smart contract from outside world. [7]



1.11 Critical issues

Blockchain is a one of the last new technology innovation and it is not clear what is your real value. As new technology trend, there are some challenges that blockchain has to overcome for becoming a real innovation today. It has several major technical barriers that could be make them impractical.

1.11.1 Scalability

Decentralized Consensus protocol, as we have seen previous, has a crucial role in the blockchain because ensures security and political neutrality removing the central party authority. Every single node on the network is responsible for securing the system by processing every transaction and maintaining a copy of the entire state. All of this comes at the cost of scalability because decentralized by definition limits the node to process an limited amount of transactions. It implies a *low throughput*, blockchain can process only a limited number of transaction, and *slow transaction times*, the time required to process a block of transactions is slow, for example in Bitcoin this time is 10 minutes. As consequence, growing up of the network will bring an inefficiency because there will be more transactions to validate but the low throughput is steady.

1.11.2 Privacy

Public blockchain is completely transparent because all data are stored within the ledger which is distributed along the network. We have seen that each transaction is referred to a address comprised solely of numbers and letters. It could be appear that the transactions are anonymous and anyone can associate them to an real identity. However, it's not really that. If someone

makes the connection between the pseudonym and a real identity the secret is revealed and the person lost his or her privacy.

Furthermore, when we use a blockchain where smart contracts are stored, it provides a lot information about user such as senders and recipients or transaction data itself. Unauthorized users, hackers and competitors could view data and retrieving important information for own aims.

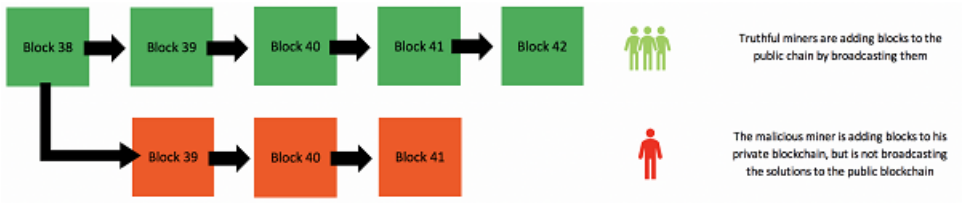
1.11.3 Double-spend attack

Double-spend or 51% attack is a vulnerability which affect Proof-of-Work consensus. It consists of trying to spend cryptos on blockchain twice by miner or group of miners.

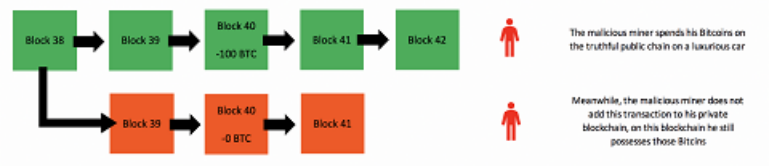
For example in Bitcoin Blockchain, a miner Alice spend 1 BTC on a computer. Her Bitcoin is transferred from her to company and she will get the computer. By performing a double-spend attack, miner Alice can now try to reverse that Bitcoin transfer which it permits her to possess both the computer and her Bitcoin, allowing her to spend that Bitcoin again.

This type of attack can conduct when a miner or group of miners controlling more than 50% of a network's mining power, 51% is enough. It permits to find the hash solution more quickly than other miners and so creating a chain longer.

Corrupted miner creates blocks in parallel with other miners but he does not broadcast their block to the network. In this way there will be two version of blockchain, one version corrupted which is malicious miner working on and one that is being followed by uncorrupted miners.

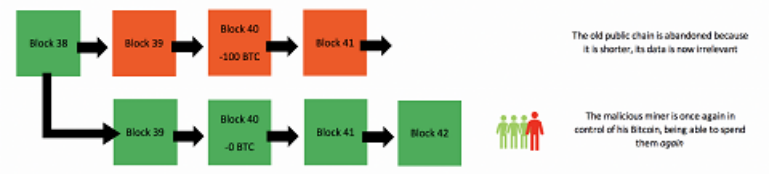


While corrupted miner are mining new block on his chain, he will spend some Bitcoin creating new transaction which will include on truthful version on blockchain and not in his malicious version.



The blockchain is programmed to follow a model of democratic governance. Miners when receive more than one block referred to different blockchain they decide on depending which is longest. Having more than 50% of a network's mining power permits malicious miner to find the hash solution more quickly than other miners and so creating a chain longer.

As soon as the corrupted miner achieves a longer blockchain, he broadcasts this version of the blockchain to the rest of the network. Other miners will follow the longest chain switching own version to the corrupted received by corrupted miner.



In conclusion, corrupted version of blockchain will be considered the truthful blockchain and transactions that are not included on this chain will be reversed immediately. It is called double-spend because miner has spent his Bitcoin thought a transaction but it will be reverse and miner can spent these Bitcoin again.

1.11.4 Formal contract verification

An unsolved problem in the world of smart contracts is the "Formal verification". It is comparable to a “formal proof” in mathematics where using the foundational axioms of mathematics and primitive inference rules it is checked by computer. Formal verification is in relation to a software program and it consists in a methodology to verify if the program, with certain input data, follows specification giving correct data output. In other words, we first state an invariant about the program, and then we are obliged to prove or disprove that statement.

One example of a specification language is Isabelle, which is a generic proof assistant that allows mathematical formulas to be expressed in a formal language and provides tools for proving those formulas in a logical calculus.

Formal verification for programs encoded within smart contracts is crucial. Immutability is the main feature of a smart contract and you can’t update or fix them once they have been deployed onto the network. It means that before to use those contracts in real-world applications it’s needed that everything is right. Moreover, smart contracts are public and data stored within smart contracts is open for anyone to view. While this provides openness and transparency, it also makes smart contracts very attractive targets for hackers.

1.11.5 Storage constraints

One advantage of public blockchain is to able to maintain all data writing on it indefinitely. It means that every information which wants to store will be append on blockchain with previous data. More users exchange data and bigger will be the size of the blockchain ledger. Moreover, according to feature of blockchain, every new information will be diffused among the network so that all data remain on every node.

As a result, storing information on a public blockchain database means that:

- every full node in the network stored data.
- blockchain is immutable and store data indefinitely.

Therefore, these features impose a huge cost on a decentralized network where every full node has to store more and more data into infinity. It causes that for any realistic application that gets built on the blockchain remains a huge handicap.

1.11.6 Quantum computing threat

Most dangerous threat to cryptography and in consequence to cryptocurrency is the issue of quantum computers.

In the future, large quantum computer could be efficiently broken public-key algorithms. Nowadays, quantum computers are still somewhat limited in what types of problems they can solve but it won't always be that way.

It's important that as we design and build the blockchain and the cryptography that underlies it, we need to be thinking about how to make these properties quantum-proof.

1.11.7 Others

I explain the main disadvantage of blockchain but there are others equally important such as to have a reliable and efficiently consensus mechanism or missed of central authority that while on one hand, this affords us the dream we all are after — a completely trustless, open, and permissionless system — on the other hand, there literally is no safe upgrade path for the protocol, and no one responsible for setting and maintaining standards.

Chapter 2

Bitcoin

2.1 Concepts

2.1.1 Overview

The first implementation of blockchain technology is Bitcoin and it is useful to have a look it to understand how this invention works and how it was developed.

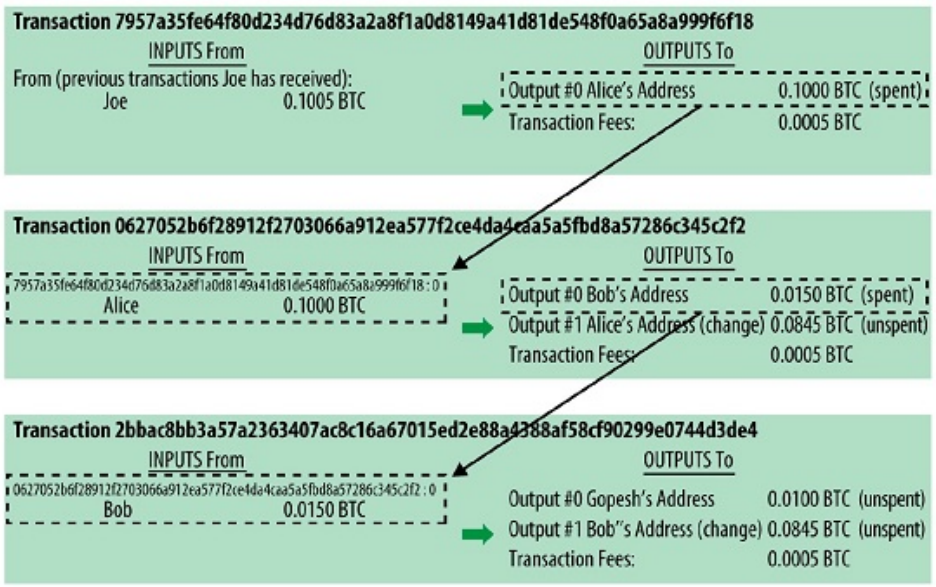
Bitcoin is a powerful blockchain application and consist in a digital money ecosystem where you can buy and use an asset electronically. As a traditional currency, it can be exchanged among participants in the Bitcoin network to transfer value from sender to recipient directly, rather than having to be mediated through financial institutions.

Users of Bitcoin own keys which need to sign transactions to unlock the value and spend it. These keys are the unique prove of ownership of Bitcoin in the network, if you lose them you lose the coins. Therefore, it is crucial to store them in a safe place as digital wallet on each user's computer or smart-phone or external hardware.

Mining is the process that takes care of creating Bitcoin and consist in a competition among "miners" to find solutions to a mathematical problem while validating Bitcoin transactions. Any participant in the Bitcoin network can be a miner, using their computer's processing power to verify and record transactions.

2.1.2 Transaction

In Bitcoin blockchain, a transaction has one or more *inputs* and one or more *outputs*. Input is like a debit against a Bitcoin account and it is a reference to a transaction's output. On the other hand, outputs are like credits adding an amount to a new owner's Bitcoin address.



example of transaction in Bitcoin blockchain

In simple terms, a transaction tells the network that the owner of some bitcoin value has authorized the transfer of that value to another owner. The new owner can now spend the bitcoin by creating another transaction that authorizes transfer to another owner, and so on, in a chain of ownership. The value moves from transactions input to transactions output.

When transactions are created, outputs are slightly less than inputs because it is payed a small transaction fee to the miner who includes the transaction in the ledger. First transaction is a special transaction called *coinbase* which has only one output and it represents miner's reward for the mining effort.

Once transaction created, it is sent to node's neighbor forwarding across the entire bitcoin network. However, before propagating transactions, a bitcoin node will first validate them. Indeed, only valid transactions are propagated over the network while the invalid will be discarded. That just validated will add in *transaction pool* (*memory pool*) which contains a pool of

valid *unconfirmed* transactions.

2.1.3 Proof-of-work

As can be seen, Mining is the process to achieve a consensus on decentralized network. It consists in a competition among miners which ends with the propagation of a new block and the beginning of the next round. This new block is valid because miner has founded the solution for the Proof-of-Work (PoW) algorithm. PoW is the algorithm used in the bitcoin blockchain to achieve the consensus which takes advantage of hashes function to implement own mechanism.

A hash function gets arbitrary data in input and convert it into a fixed-length output data. If we modify this data any way and the hash re-run, a new output data is produced, so there is no way to change the data to determinate the result in advance. It means that the only way to find a particular solution is to try again and again modifying randomly the input data.

PoW algorithm means that it must give a proof of the work performed. In bitcoin blockchain, the proof is a value, which does not exceed a certain value called *target*, founded by performed more time an hash SHA-256 function which represents the work.

The process of founding this value consists that a miner build a candidate block with some transactions. Moreover, the candidate block contains a particular transaction called *Coinbase* which is used by miners to collect any transaction fees and block reward. It is the first transaction into the block and the only difference with others is that it has a single blank input. Next, the miner applies the hash function on the block header and than he calculates the hash of block header's hash plus a *nonce*. This final step is

executed more time until the result obtained from the process is less than the *target* threshold. In the last step, the nonce is the part of the input that change on every iteration.

When the target is high, the difficulty to find a hash that is below the target is less. Consequently, a lower target means it is more difficult to find a hash below the target. Its value is crucial to balance the network because a bitcoin's feature is that a block must be generated every 10 minutes, on average. This time must be constant and adjusting target value periodically is vital to maintain it steady. [8] In simple terms, if a block is founded faster than every 10 minutes, the target increases while it will be decreased if it is slower than expected.

Difficulty depends on the the global hash-rate of the network. It is calculated by the current difficulty, expected rate of finding a block and the actual rate of finding a block in the last 2016 blocks. It could be summarize in this equation

New Target = Old Target * (Actual Time of Last 2016 Blocks / 20160 minutes)

New target compared with miner's hash-rate gives information to a miner how much his probability to find blocks. Miner's hash-rate is the capacity to calculate hash in a time lapse of miner's hardware, for example 1 TH/s is 1,000,000,000,000 (one trillion) hashes per second. [3] It is possible calculate the expected value of mined blocks by miner in 24 hours. Dividing difficulty by miner's hash-rate, you have the average time required to solve a block. Considering that in one day there are 86400 seconds, dividing this number by previous you obtain the value wanted.

Once a miner found the right result of the process just explained, he broadcasts the new block to all its neighbor which will validate and than propagate the block on entire network. The block received will be add to own copy of blockchain by each node. Immediately, mining nodes stop to find the block for the same height and start computing a new block using last block received as the "parent".

2.1.4 Assembling chain

The last step in bitcoin's decentralized consensus mechanism is to add the blocks received into chains and the selection of the chain with the most Proof-of-Work. We are talking about more chains because a node maintains two chain: the main chain and another that form branches off the main blockchain (secondary chains). A branch of the main chain is used for future reference.

When a miner receives a block, there are three possible scenarios that can happen:

- A block received to a miner has a parent block which is known and it is at the top of miner's main chain. In this case, at first the miner verifies that and than it will slot the block into its main chain.

- Parent of the received block is known but it has not the same hash of the block which is at the top of miner's main chain. This block is a *stale* block for the miner. A stale block is a block that can not attach to the main chain but it could take a part in the secondary chain. It means that will be created a branch of the main chain, a secondary chain, starting from the parent block present into the main chain. If already exists a secondary chain and its last block is the parent block of the block received, the latter will be added on top of last block extending

the alternative chain. [9]

- Orphaned block is a block just arrived but its previous hash is not known. They usually occur when more blocks which were mined within a short time and the child block arrives before the parent. Orphan blocks are maintained in a set where stay until its parent is not resolved. When a miner identifies a block as orphaned, he asks to neighbors if they know its parent sending the parent hash. Neighbors which know the parent block send it to miner who verifies that this block exists into its chain. This process continues until orphaned block's chain are resolved and the miner has extended main or secondary chain attaching it.

As mentioned before, nodes maintains a main chain and a secondary chain. In the latter two scenarios these data structures could switch each other for the following principle. The main chain is the chain that must have the most cumulative Proof-of-Work associated with it. It means that it has the longest length chains and it contains the most blocks in it proving that it has worked most. When a new block is received, it could be not attached to the main chain but it could be extended the secondary chains which will be longest than the first. If it happens, the node switches the secondary chain into the main because it has more cumulative work than other chain. A miner who will start mining the next block will construct the candidate block extending this new chain "voting" with his mining power the consensus.

2.1.5 Protocol

Bitcoin blockchain implements its feature though intensive communications between peers of the network. These communications are done following a

protocol which consists on exchange of particular messages.

Typical Bitcoin protocol messages are [10]:

- version - Exchanged during first connecting, provide information about version of protocol.
- verack - Response to a version message, it informs sender to that we are willing to connect.
- addr - Set of known IP addresses and ports of the network .
- inv - Advertise its knowledge of blocks or transactions.
- getdata - Send to retrieve a block or transaction by hash.
- getblocks - Return an inv containing all blocks in a range.
- getheaders - Return a headers message containing all headers of blocks in a range.
- tx - Response to a getdata request. It sends a transaction
- block - Response to a getdata request. It sends a block
- headers - Response to a getheaders message. It sends up block headers downloading the headers of blocks instead of entire blocks.
- getaddr - Request information about known-active peers in the network. The response is an addr message.

Every message has a certain importance in Bitcoin protocol and each of them are used on a specific scenario.

Connection

When a peer wants to connect to another, first it sends a version message containing own version number, current time and block count. If the remote peer is accepting connections, he sends back a verack message and his own version message. Received this message, the peer responds with own verack if he is accepting connections from his version. Then, he exchanges getaddr and addr messages which are able to know new address in the network, addr messages could contain one or more address.

Standard relaying

An inv is the message that introduces new transaction in the network. This message informs peer's neighbors about the transaction on request to the full transaction through getdata message. Once received, the peers verify the transaction and if it is valid they will broadcast it to all of their neighbors with an another inv. However, peers who have already new transactions will not ask for. Similarly, only peer who never broadcast a transaction will send it to the network. It works the same when a peer finds new block. When someone wants to broadcast new block, he creates an inv message containing it and as above he sends it to all of their neighbors.

These mechanisms of broadcast among peers is possible because bitcoin protocol provides particular messages for monitoring the neighbors. These messages allow to each peer to have a clear picture of which IPs are connected to the network at a certain moment. In particular, every 24 hours each peer broadcasts an addr containing their own IP. These addr messages are relayed to a few of own peers and store the address if it is new to them.

Initial block download

Once a peer has connected to another, he sends a getblocks message which contains the hash of the latest block that he knows. The peer who receives this message verifies if this block is really the lastest and if is not, he send an inv message which contains up to 500 blocks ahead of the one he listed. Then, peer requests all of these blocks with getdata and the remote peer will send them to him with block messages. In this way, all of these blocks are downloaded and processed until the peer does not have all of the blocks, exchange of these message is repeated more time.

2.2 API for working with Bitcoin Blockchain

Before design and implementing the blockchain simulator, we propose the goal of monitoring Bitcoin blockchain. In this section, we look at which API we used to help us to understand better how all actors within blockchain work: BitcoinJ, Bitnodes and Blockchain.com.

2.2.1 BitcoinJ

BitcoinJ is a open source Java-based library which provides functionality for working with Bitcoin network. As its website say, it can maintain a wallet, send/receive transactions without needing a local copy of Bitcoin Core and has many other advanced features. [11] It is one of the first library which permits strong interaction with Bitcoin protocol and so implementing complex Java applications that interact with the Bitcoin blockchain.

By using BitcoinJ API, you can create a Bitcoin address on wallet saving

it to disk or retrieving the genesis block or establishing connections to the Bitcoin test network. According to be open source project, its code can check out easily from github repository into Eclipse giving a contribute to maintain the code or modifying for own purposes.

The main entities of BitcoinJ framework are:

- *NetworkParameters* selects the network which will work on. Network can be *production* or *test* where the first is the bitcoin network production while the latter is a test network which permits to experiment new features.
- *ECKeys* and *Wallet* permit respectively to create a Bitcoin address following elliptic curve cryptography(ECC) standard and storing the address and other data into a Wallet.
- *PeerGroup* is vital to manage the network connections investigating about neighbors and synchronization of the network.
- *BlockChain* handles the global data structure which Bitcoin network work on.
- *BlockStore* stores the data structure on somewhere, for example on a database (Postgres) or in main-memory.

Our first experiment with these objects was to create a *Peer Listener*. It connects to a set of peers listening about new connections or disconnections and new block which arrives and observing how many neighbors each peer has. When one of them happened, we stored the data event on a file or a database which helped us to evaluate the behaviours of the nodes into the network.

As we have seen, PeerGroup is the entity which permits to manage connections to peers. According to choice which networks we want to use and where stores blockchain which will download, a PeerGroup needs a way to discover the peers inside the network and a limit of max connections which will create. In our case, we added a DNSDiscovery and set max connections as system argument.

```
/* start monitoring blockchain */
final NetworkParameters params = MainNetParams.get();
BlockStore blockStore = new MemoryBlockStore(params);

BlockChain chain = new BlockChain(params, blockStore);

/* create peer group */
PeerGroup peerGroup = new PeerGroup(params, chain);
peerGroup.addPeerDiscovery(new DnsDiscovery(params));
peerGroup.setUserAgent("ThesisMonitor", "1.0");
peerGroup.setMaxConnections(Integer.parseInt(args[0]));
```

Once the DNS discovery on the peer group started, we added a set of event listeners required to observe our goals. All listeners were added after a peer connected to our node including the peer disconnected event listener. In both events we wrote node's ip address to which connect/disconnect and the time when it happened.

```
peerGroup.start();

peerGroup.addConnectedEventListener(new PeerConnectedEventListener() {

    @Override
    public void onPeerConnected(Peer peer, int peerCount) {

        peer.addDisconnectedEventListener(new PeerDisconnectedEventListener() {

            @Override
            public void onPeerDisconnected(Peer peer, int peerCount) {
                writer.writeConnections(peer.getAddress().getAddr().getHostAddress(), "disconnected", Utils.currentTimeSeconds());
            }
        });

        /* write connection */
        writer.writeConnections(peer.getAddress().getAddr().getHostAddress(), "connected", Utils.currentTimeSeconds());
```

The followed picture shows another listener about new received block.

```
/* set parameters for download headers */
peer.setDownloadParameters(Utils.currentTimeSeconds(), false);

peer.addBlocksDownloadedEventListener(new BlocksDownloadedEventListener() {

    @Override
    public void onBlocksDownloaded(Peer peer, Block block, FilteredBlock filteredBlock, int blocksLeft) {

        /* print peer - hash */

        writer.writeHashes(
                peer.getAddress().getAddr().getHostAddress(),
                block.getHash().toString(),
                Utils.currentTimeSeconds(),
                currentTime < block.getTimeSeconds()
            );

    }
});
```

Last listener added was for counting number of neighbors belong to every peer. It was done easily by exploiting the *getAddr* function provided on a Peer object. It returns a list of *AddressMessage* that contains neighbor's host address.

```
CompletableFuture.runAsync(() -> {

    ListenableFuture<AddressMessage> neighboorsFuture = peer.getAddr();
    try {
        AddressMessage message = neighboorsFuture.get();
        for(PeerAddress peerAddress:message.getAddresses()) {
            writer.writeNeighboors(peer.getAddress().getAddr().getHostAddress(), peerAddress.getAddr().getHostAddress());
        }

    } catch (InterruptedException | ExecutionException e) {
        e.printStackTrace();
    }
});
```

Analyzing data which we stored, we estimated that the connections among peer are not always permanent but some of them often go down. Although we observed that the blocks arrive at peers about at the same time, some peer's disconnections could cause some slight delay on receive new block.

Another experiment performed was to download entire blockchain from the network identifying from each block who was the miner.

To do that, we connected to one peer and we added a BlocksDownloadedEvent listener. After our client was connected to unique peer, we started

to download entire blockchain from it. Identifying miner's block means detect a transaction that has first input which is not a reference to an output of another transaction. BitcoinJ provides Transaction object which contains a list of TransactionInput and TransactionOutput. Verifying the outpoint of first transaction input permits to identify miner's transaction and retrieving the only transaction output from it, we obtained miner address and block's reward.

```
ListenableFuture<List<Peer>> future = peerGroup.waitForPeers(1);
List<Peer> peers = future.get();
Peer peer = peers.get(0);

peer.addBlocksDownloadedEventListener(new BlocksDownloadedEventListener() {

    @Override
    public void onBlocksDownloaded(Peer peer, Block block, FilteredBlock filteredBlock, int blocksLeft) {

        List<Transaction> transactions = block.getTransactions();

        for(Transaction tx:transactions) {

            if(tx.getInputs().size() == 1) {
                /* only one input, could be miner's transaction */
                /* verify input */
                TransactionInput txi = tx.getInputs().get(0);
                if( txi.getOutpoint().getHash().toString().equals("0000000000000000000000000000000000000000000000000000000000000000")) {
                    /* input null, it is miner's transaction */
                    HEIGHT++;

                    String wallet = "";
                    String amount = "";
                    for(TransactionOutput out : tx.getOutputs()){
                        wallet = out.getScriptPubKey().getToAddress(params).toString();
                        amount = out.getValue().toString();
                    }

                    try {
                        csvPrinter.printRecord(Arrays.asList(block.getHashAsString(), wallet, block.getTimeSeconds(), amount));
                        csvPrinter.flush();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            }
        }
```

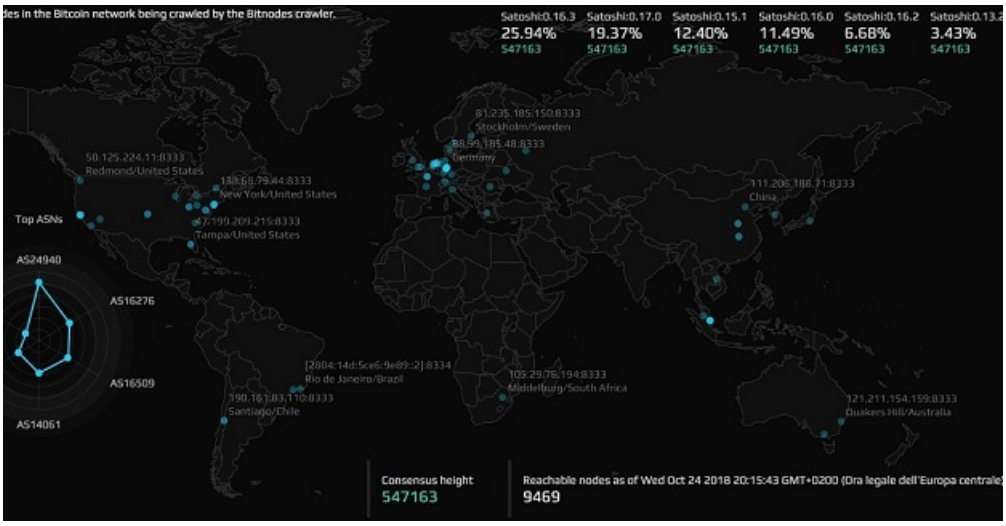
2.2.2 Bitnodes

Bitnodes is a open-source Python library developed with the goal of estimating the size of the Bitcoin network by finding all the reachable nodes in the network. [12]

Starting from a set of seed nodes, Bitnodes sends a series of getaddr messages for reaching all peers in the network. It involves only nodes running Bitcoin protocol version 70001 while nodes which runs an older protocol version will be skipped.

Bitnodes implements all Bitcoin protocol messages and so provides functions which permit to interact with nodes of the Bitcoin network. It uses Redis, an in-memory key-value store, where the project's scripts write values as key-value data which are crucial to maintain information about network.

The main feature of this library is to able on creating a clear picture of the network as we can see from image below.



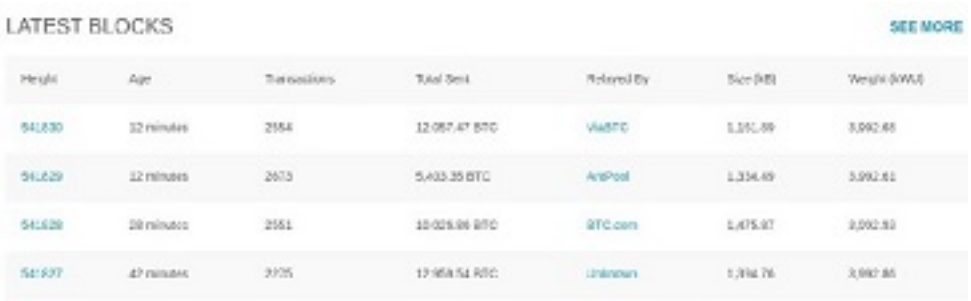
Moreover, Bitnodes provides a list of API useful for obtain a lot information about peers such as status or node's bitcoin address or a list of nodes. These API are implemented by REST services and they return JSON objects.

Exploiting protocol.py file, we used this library to understand how Bitcoin network really works. In particular, we used getaddr and addr function respectively to send requests about known-active peers in the network and to retrieve IP address and ports of peers. These features are allowed us to

build a small part of the bitcoin network evaluating for particular nodes how many neighbors they have.

2.2.3 Blockchain.com

Blockchain.com, known also as Blockchain.info, is one of the most website visited in the crypto landscape. It is a private website which belongs to a company called "Blockchain" established in Luxembourg. At first it was launched as provider of Bitcoin data but now the company also offers cryptocurrency wallets supporting Bitcoin, Bitcoin Cash (BCH) and Ethereum (ETH). Moreover, Blockchain.com provides also market information, statistics, data chats and a Bitcoin blockchain explorer which includes a complete view about last block inserted into Bitcoin blockchain looking the frequency in minutes of them.



Blockchain.com APIs are another interested services because permits easily to create a full Bitcoin application. [13] There are available as library for various languages such as Java, Python or Node or through Websocket, a low-latency channel on stream sockets. It means that by subscribing to the channel it will stay on listen to receive real-Time notifications about data when some events happen.

These are different API which are divided in four section:

- *Receive Payments* consists of simple HTTP GET requests which permits websites to receive Bitcoin payments easily

- *Blockchain Wallet* permits to send and receive payments from Bitcoin wallet but it is needed to install local a small service which be responsible for managing your wallet which will interact with this service locally via HTTP API.

- *Blockchain Data* offers services for querying JSON data about blocks and transactions. This data can be retrieved also by Websocket

- *Exchange Rates* provides information about market prices and currency data from the main Bitcoin exchanges.

We were interested in API which could permit to observe when new blocks were mined into Bitcoin blockchain therefore we exploit some feature of Blockchain Data API. In particular, we were focused on WebSocket API subscribing to a service which sends notifications when a new block is mined. The main purpose was to evaluate if notifications received were faster than BitcoinJ library explained before.

To perform a WebSocket API we exploited from Blockchain.com WebSocket API echotest at "http://websocket.org/echo.html". We connected to connection URL "wss://ws.blockchain.info/inv" and subscribed to the channel by sending the JSON message {"op":"blocks_sub"}. Once subscribed we were receiving data each time a new block was found.

As we can see, JSON object received contains all important information about new block. In conclusion, data received from eBlockchain.com WebSocket API is resulted faster of some seconds than BitcoinJ library.

Chapter 3

The Simulator

3.1 Introduction

In this chapter I focus on an implementation of concepts discussed in previous chapters, a Blockchain Simulator. The design of the blockchain simulator has evolved over all period of study about theoretical and practical aspects. Primary contribute was the study on some library discussed previously about Bitcoin world such as BitcoinJ, Bitnodes and Blockchain.info. These frameworks and tools has highlighted in more detailed what our simulator needed and its requirements described in the next section.

After defined the requirements, we have to choose which programming language used. The choice was Python because we exploited an existed library which was able to create a graph among nodes and distribute an object over the network. We used this code as base of simulator and evolved it with all features that I will explain later. The software was developed following an Object-oriented programming without using any particular framework or complex design pattern.

3.2 Design

The purpose of building a simulator was to create a test environment that can represent a miner ecosystem where we can test different methods to executing on a live blockchain.

A simulator must accept input parameters through which it is possible change behaviour of the system. It is crucial to distinguish between input parameters of the network with those of the each miner. While the first determine in which condition the system and all miners works, miner parameters define what is the own strategy within the network. Changing these parameter we will have different outputs which permit to evaluate the impacts of

diverse strategies.

Requirements system were defined based on all study discussed in previous chapters but they were modified during developed steps. Primary goal of design was to identify autonomous entities and their fields:

- *Ecosystem* represents the ecosystem within the entire network works. Its main attributes are *n-node* and *n-iteration*. First indicates how many nodes must have the system while *n-iteration* is the number of block that will be mined during execution of the ecosystem. *wait* is an another significant attribute because it establishes how long the network must wait to mine next block since it has mined last block.

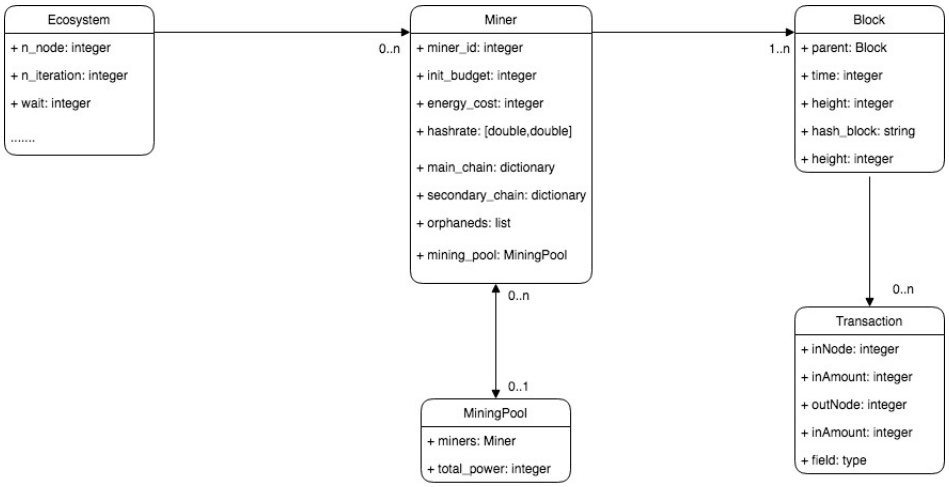
- *Miner* corresponds to a particular node inside the blockchain world. It has a *initial budget*, a *energy cost* that spent for mining, an *hashrate* and he can belong to a Mining Pool. In addition, he maintains three type of chains: the main chain, which extends with blocks that he mines, the secondary chain, a branch of the main chain, and orphaned set, block without parent that he has to resolve at the end of the iteration.

- *MiningPool* contains a list of miners and a total power which is a sum of miner's computational power which belongs to this mining pool. This field is useful when the system establishes the amount of reward for each miner.

- *Block* represents a block and it has a *parent block*, a *time* which is when it is created, the *height* of the block in the blockchain, an *hash block* calculated when the block is mined based on previous information. Besides it has a list of transactions where the first is the coinbase transaction which indicates who is the miner that has mined this block.

- *Transaction* has *inNode* and *inAmount* which are respectively miner who are sending the coin and amount of coin to send. The are the same attribute about who receives the coin, *outNode*, and amount of coin received, *outAmnout*. *fee* field is the fee which miner will receive from transaction. The entity was designed thinking about the purpose of the simulator but it does not actually reflect a real transaction in Bitcoin world.

Flow chart in below image illustrates the fully scheme about entities and how they relate to each other within a system.



3.3 Use Cases

Before implementing the simulator it was useful writing some use cases to understand if the system requirements defined met the needs of mine. Use cases helped us to capture how entities will interact each other to achieve a

specific goal. By them, I described the step by step process how complete a certain goal and which were actors' role within the process.

3.3.1 Mining block

First use case designated was the process of mining new block. As in really world happen, miner's probability to find a new block is given by his hash rate.

One way to represent the hash rate is though a n intervals of numbers dividing a certain unit, for example 1. When Ecosystem will create, it will divide randomly 0-1 unit in n intervals assigning each of them to each miner. For example, a miner will have 0,12-0,18 range or another will have 0,68-0,81. There will be miners with an hash rate high and others with lower like really happen in Bitcoin network.

Block mined will emulate extracting one number among 0-1 unit. The miner who has the interval which includes the number extracted, will be the miner of the new block.

3.3.2 Add block to blockchain

Another important use case defined was about how miners add a block into own blockchain. Miner of a new block manages it differently rather than other nodes because while the first adds it directly to own main chain, a node who is not the miner before checks if block's parent is on the top of own main chain.

The first one does not verify none because he builds new block putting as new block's parent the last block added on own main chain extending the latter one.

Nodes who are not new block's miner have to check if they can extend own main chain with this block. They will verify if the block that stays on the top of own main chain and block's parent are the same. If that is true, these nodes will grow up own main chain with new block, otherwise they must handle it in different way.

3.3.3 Stale block

When a miner can not extend own main chain with block received, this block is called *Stale* for the miner. It is identified as stale when new block has the parent different than last block which is on top of miner's main chain.

This type of block causes the creation of a secondary chain which will be a branch of the main chain where last one o more blocks are different. The secondary chain must be maintained to the miner because next blocks that arrive may extend it rather the main chain.

Now that exists two chains, miner will check for next blocks if they will take part of the main chain or if they will be attached to the secondary chain. Each time the latter scenario will happen, miner will have to do one more action. It consist to verify which chains has the most accumulated proof-of-work, in simple terms what is the longest chain. In case the secondary chain is the longest, miner must switch current secondary as main chain.

3.3.4 Orphaned block

Exists another last scenario to handle during process of assembling the chain. It occurs when new block can not extend either main chain and secondary chain. Orphaned block is a new block that has a unknown parent for the miner because it does not stay on none of miner's chains.

Orphaned block is added by miner to a list and they will be resolved at the end of the one iteration in the same time of when block arrived.

Resolving an orphaned block means that the miner will ask recursively to all neighbors if they know block's parent until he attach it to main or secondary chain.

Once finally block is resolved, the miner will handle of choice which chains must consider as main between current main and secondary chain like explain in previous section.

3.3.5 Miner budget

One fundamental information that simulator must give back is the current budget of the miner. It is important both during the simulator running and at the end of the execution.

Miner starts with an initial budget given from configuration settings. It decreases of energy cost every iteration because miner spend money for mining and it increase with the rewards when miner finds new blocks.

However, increases are not permanent because we must consider an important aspect of the blockchain. Not all blocks mined will be part of the global main chain because miner may build new block on the top of the main chain which is not coherent with the entire network.

3.4 Implementation

In this section we explain how we implemented use cases discussed before. We will show some pieces of simulator's code focusing on the main functions and explaining what is the reasoning about their implementation.

3.4.1 Initialize Ecosystem

One purpose of simulator is to enforce various behaviour patterns on the network. Behaviour is a object that represents a behaviour pattern that we want to apply to simulator. It implements a method that returns a set of initial miners which will take part of the Ecosystem. Moreover, a Behaviour must provide a function for handling miner's dead and others two method respectively that say if the simulator has to mine and has to stop.

Default class is a Behaviour object that accepts as parameter the number of network's nodes, the number of block to mine and the time of waiting for the next block to mine. After setting constant values such as energy cost or budget, it creates n miners assigning as hash-rate a random interval.

WaitUntilFullBroadcast is an another Behaviour object which implements a pattern where next block will be mined only after previous block has achieved all nodes of the network while ValueChange is a behaviour that sets miner's values such as budget and hash-rate randomly also when a new miner is introduced into the network.

```
def _init_miners(self):
    probs = list(map(lambda x: random.random(), range(self.n_node)))
    tot = sum(probs)
    probs = map(lambda x: x/tot, probs)

    self.miners[0] = Miner(0, [0, probs[0]], self.energy_cost, self.budget, self.behaviour, self.genesys_block)

    for i in range(1, len(probs)):
        old_val = self.miners[i - 1].get_hashrate()[1]
        new_val = old_val + probs[i]
        self.miners[i] = Miner(i, [old_val, new_val], self.energy_cost, self.budget, self.behaviour, self.genesys_block
```

Once Ecosystem has obtained miners from Behaviour object, it creates the erdos renyi graph using the set of nodes and p where p is the probability for edge creation. Erdos Reny is a model for generating model graphs where nodes have the same probability to have neighbors. Graph's nodes represent the miners into the network while graph's edges are the connections between

miners.

3.4.2 Run method

An Ecosystem with a certain behaviour is ready to work. Calling run method, it starts the operations of mining and propagation of blocks over the network. The following code is contained inside run function.

```
while not self.behaviour.is_time_to_stop(self.blocks):

    new_blocks = []

    if self.behaviour.is_time_to_mine(self.blocks):

        rate = random.uniform(0, 1)

        # check if some miners have mined blocks and add them in new_blocks
        for key in self.miners:
            miner = self.miners[key]
            blocks_miner = miner.mining(rate, self.time, self.reward, 1)
            new_blocks.extend(blocks_miner)

        if len(new_blocks) > 0:
            # add new_blocks inside blocks_to_broadcast
            self.setBlocksMined(new_blocks)

    self._broadcast_one(print_graph)

    if print_graph: self._plot_img()

    for key in list(self.miners):
        miner = self.miners[key]
        self._printBlockchains(miner)
        self._resolveOrphaned(miner)

        if miner.getBudget() < 0:
            self.graph.remove_node(miner.getId())
            new_miners = self.behaviour.miner_dead(self.miners[key])
            self._add_node(new_miners)

    self.time += 1
```

First if is relative to process of mining. It extracts a random value and it passes this number to miner's mining function who return one or more blocks if he is the miner.

Next, broadcast_one propagates of one unit time the blocks which are into the network and are not already arrived to all. The propagation on a unit time consists in a crossing of an edge from a node to other.

Finally, for each miner the function prints the blockchain on log, resolve orphaned blocks and remove miner from the network if miner's budget is spent.

3.4.3 Broadcast one

Broadcast is one of the most complex function in the simulator. It propagates a set of blocks from a node to another and it happens if the node destination has not received at least one of these blocks. At the end of the broadcast, a block which has achieved all nodes will be remove from list of blocks to propagated. In case the list is empty, the function does nothing and return false informing that nothing into the network has changed.
The following code is the core of the function

```
for n in filter(lambda n: n not in active, self.nodes):
    # get n's neighbors
    neighbors = list(self.graph.neighbors(n))

    # for each block, get neighbors of n which has it
    # and what is the hash block with more neighbors of n
    neighbors_have_block,max_hash = self._getNeighborsHaveBlockAndMaxHash(n, neighbors)

    if max_hash is None: continue
    # some n's neighbors have almost one block which n does not have
    # maxHash is the hash block closest to n
    has_changed = True

    # add this block to n's blockchain
    consensus_new[max_hash].append(n)
    self.miners[n].addBlock(self.blocks[max_hash])

    del neighbors_have_block[max_hash]
    if len(neighbors_have_block) == 0: continue

    # if there are more than one block, add others blocks
    for key in neighbors_have_block:
        if len(neighbors_have_block[key]) == 0: continue
        consensus_new[key].append(n)
        self.miners[n].addBlock(self.blocks[key])
```

It iterates over the nodes not active where a node not active is a node which have almost one block that have not received. Each of these not active first gets the list of neighbors. Next, it verifies if there are some node's neighbors which have some blocks that the node has not received yet otherwise jump to next miner.

maxhash contains the block's hash which is owned by more neighbors. This block has the priority to add into miner's blockchain while other blocks will be added after it. After a miner adds the block into his blockchain, the function maintains which miner has received that block.

3.4.4 Mining

Mining is the first function that are not in Ecosystem object because it is provided by Miner object. It simulates the mining process for a miner. Miner spends money for energy that consumes during mining process so every time that this function is called we decrease the budget of energy cost. Then, function verifies if the rate extracted belong to miner's interval because it means that he is the miner of the current block. Miner creates the block including the special coinbase transaction and he extends his main chain with this block.

Next, miner adds the block to "blocks_to_release" list which will contain all blocks to shared unless he does not want to release the block immediately for some reasons.

According to a particular condition, the function ends adding the blocks which are not released in the previous iteration to "blocks_to_release" list. It is performed by each node, also who is not the current miner. In our case the condition is very simple because we only check if the current time is higher than block's time.

```
def mining(self, rate, time, reward, fee):
    self.initBudget -= self.costo_energia
    blocks_to_release = []

    # is miner of this block
    if self.hashrate[0] <= rate < self.hashrate[1]:

        # add block to the main chain and add miner's transaction
        mc_height = len(self.main_chain)
        block = Block(self.main_chain[mc_height-1], time)
        block.addTransaction(Transaction(fee, self.minerId, reward))

        self.main_chain[mc_height] = block

        if self.isShareBlock():
            blocks_to_release.append(block)
        else:
            self.blocks_to_release_after.append(block)

    # release block mined before
    for item in list(self.blocks_to_release_after):
        if time > item.getTime():
            blocks_to_release.append(item)
            self.blocks_to_release_after.remove(item)

    return blocks_to_release
```

3.4.5 Add block

Add Block function implements most activities that a miner does when he receives a new block. First, he verifies if block's parent is on the top of main chain. It means that new block must be added into main chain without do any other operation.

When it does not happen, a branch of the chain, called secondary, maintained by miner join the game.

Block will extend it in case its parent is on the top of secondary chain causing that chain is longest than the main. It means that it has the most accumulated proof-of-work and so it becomes the main chain. Otherwise, block is a orphan and it is added into orphans list which will discuss soon.

Secondary chain may not already created by miner for to be extended.

However, his creation is not so obvious because it exists if block's parent is contained into main chain in any position. A unknown block's parent does not allow making a secondary chain and this block will be added into orphans list as previous.

```
def add_block(self, block):
    # verify if parent is on top of the main chain
    if not Util.verify_belong_chain_and_add(self.main_chain, block):

        # if it's not, verify if secondary_chain exists
        if len(self.secondary_chain) > 0:

            # verify if parent is on top of the secondary chain
            if Util.verify_belong_chain_and_add(self.secondary_chain, block):

                # if it's in secondary chain, check the height of the chains
                # if secondary chain's height is higher than main chain's height
                #   switch main and fork chain
                if len(self.secondary_chain) > len(self.main_chain):
                    tmp_chain = self.secondary_chain
                    self.secondary_chain = self.main_chain
                    self.main_chain = tmp_chain

            else:
                if block.getHeight() > len(self.main_chain)-1:
                    # block is an orphan and add it inside of list orphans
                    self.orphans.append(block)

        else:
            # if it's not check if there are some parent within the main chain
            # and building secondary chain
            new_secondary_chain = Util.build_secondary_chain(block, self.main_chain)

            if new_secondary_chain is not None:
                self.secondary_chain = new_secondary_chain
                self.secondary_chain[len(self.secondary_chain)] = block
            else:
                if block.getHeight() > len(self.main_chain)-1:
                    # block is an orphan and add it inside of list orphans
                    self.orphans.append(block)
```

Orphaned blocks are maintained into orphans list and they are resolved at the end of each iteration_one function by resolve orphans block. The miner iterates over orphans list and for each orphan, he asks to his neighbors if they know the block trying to rebuild the right chain. If a neighbor has not got this orphan within chains, miner asks to next neighbor until he finds

a known parent. In case he does not resolve the orphaned chain, he discards the block and move on to the next one.

3.4.6 Miner budget

Retrieving miner's budget could be easily implemented by adding coins each time that a miner creates new block. However, it introduces a bug because a new block mined might not take part of the global main chain.

To avoid this, the global budget is divided in two part, initBudget and chainBudget. The initBudget is given by the initial budget less the energy cost which is spent over iterations. On the other hand, the chainBudget is calculated on miner's main chain iterating over blocks and transactions. It is the sum of the all OutAmount belongs to a transaction where the miner is the OutNode to which is subtracted the sum of OutAmount where the miner is an InNode.

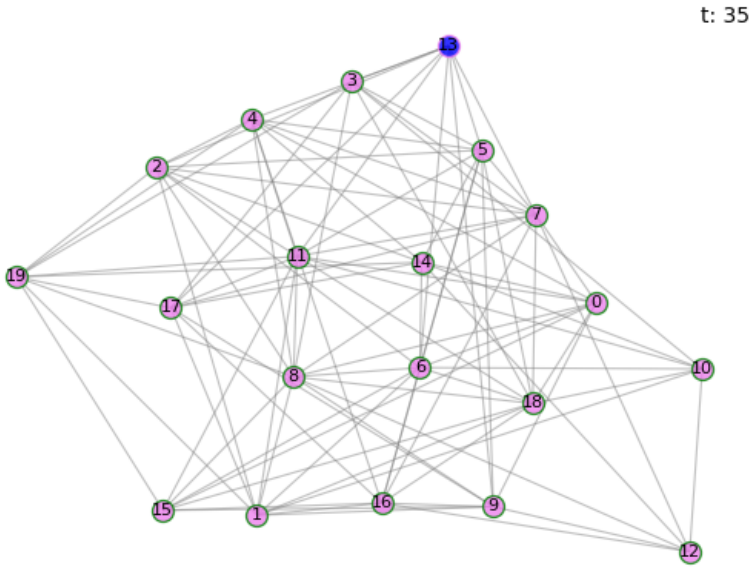
```
def get_budget(self):
    return self.initBudget + self.get_chain_budget()

def get_chain_budget(self):
    budget = 0
    for i_block in self.main_chain:
        block = self.main_chain[i_block]
        for t in block.getTransactions():
            if t.getOutNode() == self.minerId:
                budget += t.getOutAmount() + t.getFee()

            if t.getInNode() == self.minerId:
                budget -= t.getOutAmount()

    return budget
```

get_budget function return the miner's global budget.

3.4.7 Print graph

In addition to previous behaviors, we designed a function which could visualize how simulator really works. Print graph creates a series of pictures about the trend of the network over the simulation showing how blocks move among nodes. Each picture displays the state of the network in a certain instant of time t. Circles represent the miners while the edges are the connections between them. It exploits matplotlib library, in particular pyplot module, which is able to draw a graph created by networkx.

When a miner finds a new block, he is associated to a colour and miner's circle is colored with this color. Next, also neighbors which receive new block will be colored with the same color until all circles are colored. It could happen that there are new more blocks in the same iteration and in this case there will be circles colored with different colors. The followed picture is an example of picture created by our function.



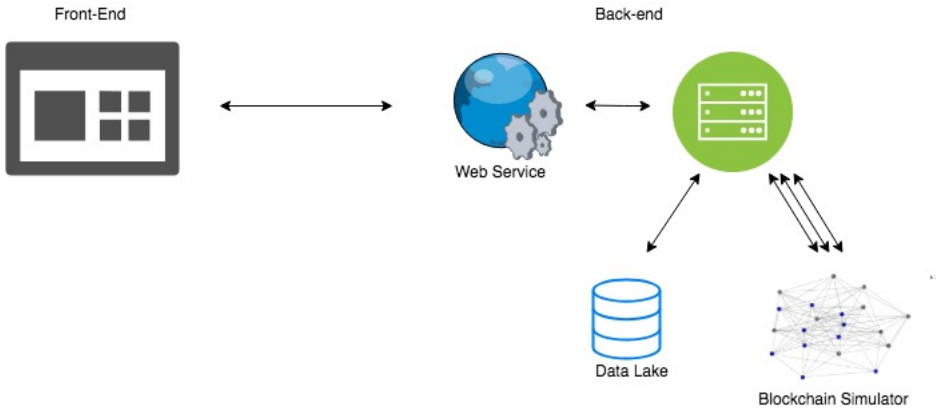
Chapter 4

Numerical Experiments

4.1 Introduction

The main goal of building a blockchain simulator was to enforce various miner behaviour patterns and analyzing the outputs. In this section we will show a series of experiments applied on the simulator and will discuss about the results.

To understand clearly the significance of outputs we decided to build a dashboard where choose the experiment to run and visualize the result by charts.



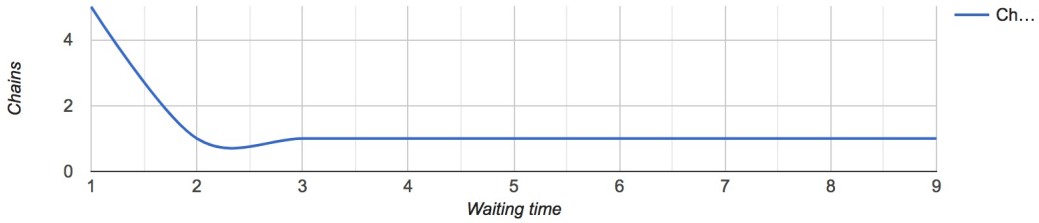
As we can see by picture, Dashboard is a web application which is divided in front-end and back-end. They communicate each other through REST services provided by back-end.

Front-end sends the inputs to back-end which executes a set of threads that runs the simulator with input received. These threads collect the simulator's outputs and store them on a Mongodb. While the threads works, front-end waits the outputs doing a polling every 30 seconds to back-end. When all threads complete their work, front-end receives the results and shows them using the Google chart tools, in particular line chart and bubble chart. All experiments were done using constant values such as number node as 20 and number iteration as 40.

4.2 First experiment

As first experiment, we decided to observe how number of branches of chains were created into network by varying the waiting time. The waiting time is the time that the network must wait before mine the next block.

We started with a waiting time of 1, that means every instant of time t there was a new block, ending with 9 t value.



The chart above shows how a small waiting time causes an high number of branches of the chain. Only after a value higher of 1 t all nodes converge in a unique main chain.

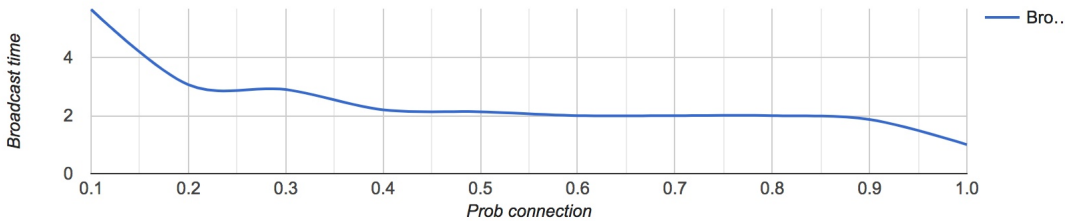
When miners start generating new blocks within a period of time shorter than network latency, different parts of the network become inconsistent.

Moreover, this behavior introduces delay in the network because it occurs multiple interaction between nodes to synchronize their chains.

Overall, we deduct that the network needs a certain waiting time to guarantee a synchronization between nodes. The real world of Bitcoin blockchain works in the same way. A block is mined every 10 minutes to avoid more frequent forks of the main chains.

4.3 Second experiment

In the second experiment we observed by varying the connection between miners, how long a new block takes to achieve all nodes. The connection between miners into the simulator is given by the probability p for edge creation explained in previous chapter. A p value equals to 1 means that all nodes are connected to each other.



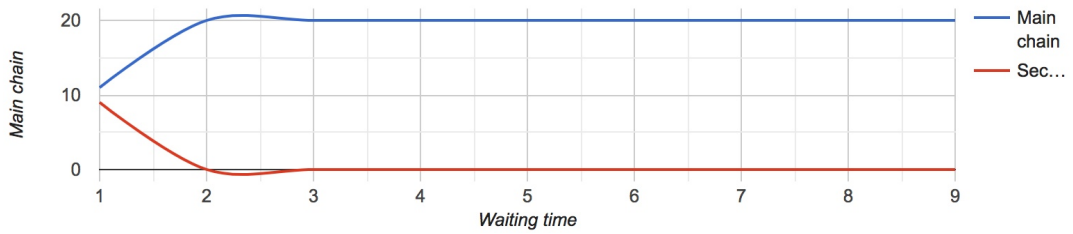
The plot starts with an high value of broadcast time because the graph has a low connectivity. That value decreases when the probability of connection between nodes is 0.30 or higher. On a strong connected graph, new blocks take only one step to achieve entire network.

This experiment shows how is crucial that the network must have an high connection between nodes. Low connectivity means delay which causes inconsistent and increasing of interactions among nodes with more consumes of resources. An high connectivity helps to maintain a good healthy of the network.

4.4 Third experiment

The third experiment is a specialization of the first. It wants to view how many nodes have the global main chain, as blue line, and how many have

secondary chains, as red line, by varying the waiting time. Also in this experiment, we started with a waiting time of 1 ending with 9 t value.



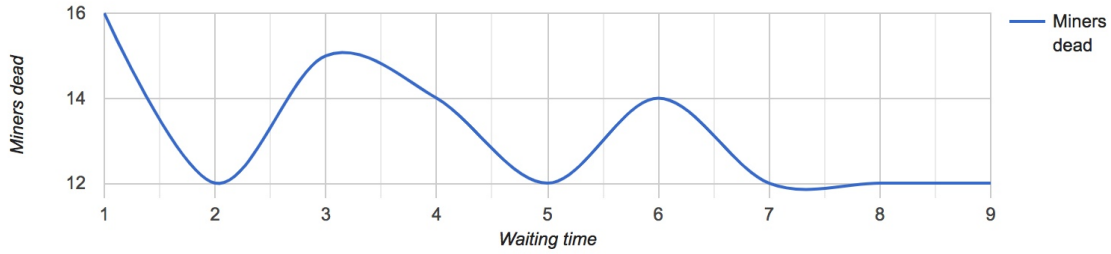
As previous experiment showed, an high number of branches of the chain occurs when the waiting time is small. In particular, this experiment highlights that with a waiting time one, the number of nodes which have got a secondary chain is higher than the number of who own the main chain.

By contrast, a waiting time value of 2 or more implies that all miners are synchronized on the same main chains and none build other branches.

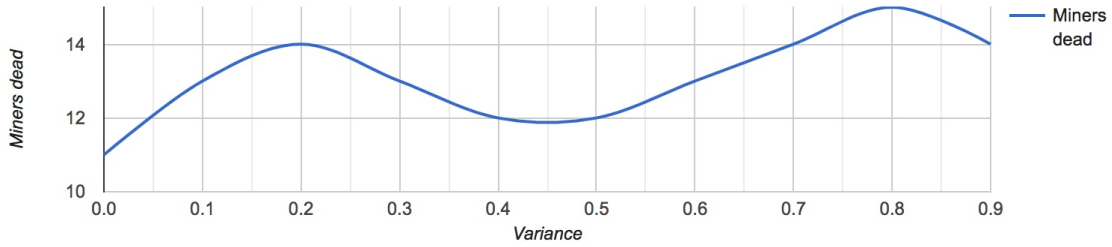
The same conclusion of the previous experiment are deducted. A good healthy of the network is guarantee by a certain waiting time.

4.5 Fourth and fifth experiment

Fourth and fifth experiments were designed to observe respectively the dead of miners by varying the waiting time and variance of miners' hash-rate. This latter value represents how the hash-rate is distributed among miners. A low value of variance means that the network is balanced because all nodes have about the same hash-rate. On the other hand, an high variance implies a set of miners which own a huge power and another set with a little power.



Experiment four shows how miners dead and waiting time are not much correlated. It occurs changes and decrease overall, only after a waiting time value of seven the line is steady.

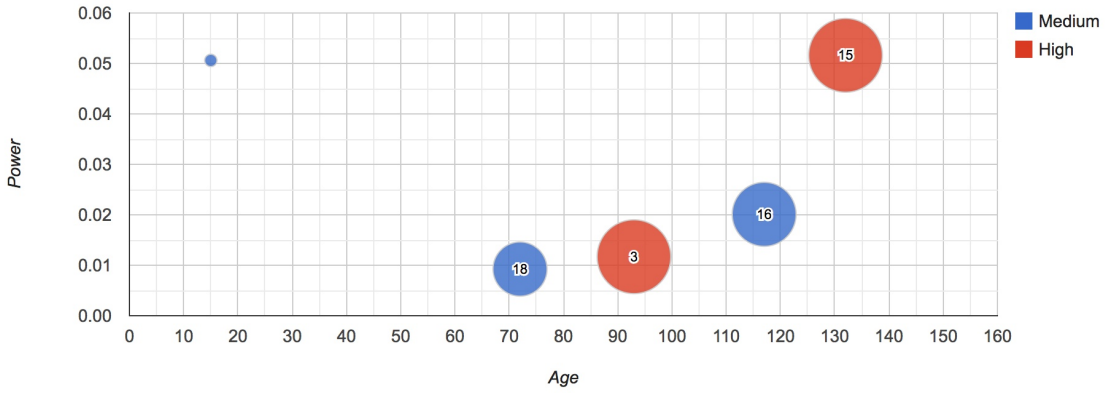


By contrast, fifth experiment gives information about relation between the features. Miners dead shows a slow and steady increase over the graph. As we expected, the dead are few when the variance is low because the network is balanced, and more dead when there are powerful miners and miners which own a little power.

4.6 Sixth experiment

Different from previous, sixth experiment examines many miner's features such as power, budget, energy cost and his age when he died where initial power, budget and energy cost are assigned randomly. It wants to provide

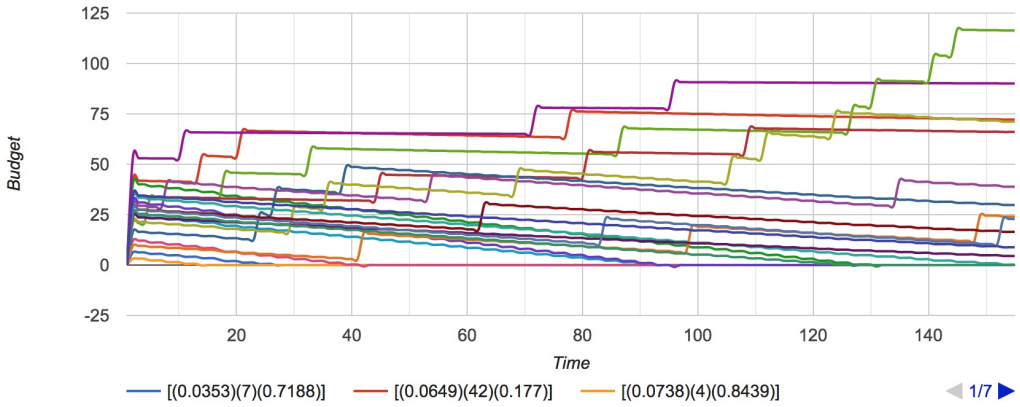
information about It wants to provide information about miner when his budget is over.



A better visualization is given by a bubble chart which adds information in terms of size compared to a line chart. In particular, a big bubble represents an high budget while the words "Medium" and "High" mean that a miner had a medium or high energy cost. This graph illustrates how the budget could be crucial because two miners which have the same power live in different way. The miner with a little budget and a low energy cost dies soon in contrast with the others that have an high energy cost and less powerful but they live more.

4.7 Seventh experiment

The last experiment observes the trend of miner's budget over the simulation. It wants to identify what is the miner's strategy that works better. A miner strategy represents the initial values of budget, hash-rate and energy cost.



The plot above visualizes for each miner how the budget changes over the time in a situation where the simulator has 20 nodes and initial values of miners were assigned randomly. The peak of 12 that we see on some lines is caused by the reward that a miner receives from the network when finds a new block. Also in this last experiment, the trend of the graph is like we expect. Miners which have an high hash-rate (6-7%) and an energy cost low (0.15-0.20 max) grows their budget over the time though they have a little budget. The same progress is performed by miners with an energy cost about 0.30 underling that an powerful hash-rate follows an income of coins. It also true that when spending on energy starts to become high such as 0.50 or 0.60 a miner goes at a loss. The lines which decrease are referred to strategies with an high energy cost or an low hash-rate. An huge budget helps only to maintain the miner alive for more time but the result is that he will dead.

Chapter 5

Conclusion

Blockchain is an disruptive technology having an huge potential and it is at the beginning of its technological evolution. In particular, Bitcoin is a powerful blockchain application and consist in a digital money ecosystem where you can buy and use an asset electronically which eliminates the need for a third party because its property permits to have a distributed trustless consensus. However, it still has some limits such as low throughput or high energy consumption which do not fit with enterprise applications.

Our work was fundamental to have a global knowledge about this emerging technology. It allowed us to understand the advantages and the disadvantages about DLT but also to realize that many aspects are yet to be discovered. These studies highlighted in more detailed what our simulator needed and its requirements. A further contribute is given by the study on some library about Bitcoin world such as BitcoinJ, Bitnodes and Blockchain.info.

The choice of Python as programming language allowed to speed up the implementation of simulator because we exploited an existed library which was able to manage a graph and propagating an object over the network.

The execution of experiments discussed in chapter 4 permitted to verify

empirically some behaviours of the network in certain initial conditions. Analyzing the outputs provided by simulator is demonstrated that while some miner behaviours are very steady, some others depend of how the network evolves in terms of miner features. In some experiments, varying input parameters as number of nodes or number of iterations does not change the trend of the results, for example in the first three experiments. On the other hand, fourth or seventh alter their outputs over more test though inputs are always the same.

Bibliography

[1] K. Wust and A. Gervais, "Do you need a blockchain?," vol. 37, p. 4, 2008.

[2] V. Buterin, "On public and private blockchains," 2017. IACR Cryptology ePrint Archive.

[3] A. M. Antonopoulos, *Mastering Bitcoin.* giu. 2017.

[4] "Iota docs." https://iota.readme.io/.

[5] V. R. P. N. N Gaur, L Desrosiers, *Hands-on Blockchain with Hyperledger.* 2018.

[6] J. Reed, *Blockchain - Smart contracts.* 2016.

[7] "Oraclize docs." http://docs.oraclize.it/home.

[8] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," vol. 37, p. 4, 2008.

[9] "Bitcoin api." https://bitcoin.org/en/bitcoin-for-developers.

[10] "Bitcoin wiki." https://en.bitcoin.it/wiki/Main$_P age$.

[11] "bitcoinj docs." https://bitcoinj.github.io/.

[12] “bitnodes api.” https://github.com/ayeowch/bitnodes.

[13] “Blockchaininfo.com.” https://www.blockchain.com/.