“ALEXANDRU IOAN CUZA” UNIVERSITY OF IAŞI

FACULTY OF COMPUTER SCIENCE

BACHELOR THESIS

Creating a Blockchain

proposed by

Iaroslav Mazur

Session: *February, 2020*

Scientific coordinator:

Andrei Arusoaie, PhD

Table of Contents

1. Introduction

At the dawn of our civilization, the human kind was - naturally - very decentralized, with people living their lives independently from each other. They were fighting for their survival in rather small groups (usually, of blood-related relatives). That was undoubtedly one of the key factors affecting every sphere of their lives; one of the fundamental parts of their essence.

Later on, people started noticing that when the number of members in a group increased, a lot of the problems they had been dealing with were if not solved completely, then, at least, reduced to a bearable level. That discovery marked the beginning of the modern world structure, becoming one of its first building blocks.

With the course of time, the overall level of decentralization has been slowly, but surely, decreasing. People have been forming bigger and bigger groups, becoming more cohesive within them, while their groups have been gradually getting more interconnected and interdependent. This has been happening both globally and - even more so – locally, with some countries, seemingly, having run to extremes.

Nowadays, centralization is such an integral part of our society that many modern people can't even imagine how we could live without (or even "with less of") it. Probably, the best proof of this is that, generally speaking, whenever people face a problem, they tend to think of some centralized way of solving it.

This way of organizing our society is, obviously, not pure evil (otherwise, we wouldn't have been supporting it for millenniums). Given the right conditions and applied to a reasonable extent, it does help. Here are just some examples of this:

- the physical proximity to a thriving ecosystem (which is a result of being a part of/contributor to it) increases the quality of one's life, while reducing its cost;
- the decision-making process becomes the responsibility of the people who are "more competent" for the task. This, in its ideal (theoretical) form, lifts the unnecessary burden from the shoulders of the rest of the population;
- living alongside people who are more talented and/or experienced in the domains that we're fairly ignorant of means a relative ease of access to the knowledge of all these people. Thanks to this, we can concentrate on doing what we, personally, are best at,

entrusting the problems in other spheres of our lives to those who are skilled-enough to solve them.

While all this sounds good in theory, the centuries-old history of applying this very theory has unveiled multiple drawbacks of living in a centralized society. Perhaps, the biggest one of them is that most of the power concentrates in the hands of a relatively small number of people, who tend to become corrupted by it.

Among other problems, are:

- the need of a high level of trust in the power-wielding authority;
- an overall inefficiency caused by the bureaucratic approach to decision-making and problem-solving;
- a lack of flexibility when a change must be made (due to the fact that only those who exercise power participate in brainstorming the topic);
- a severe vulnerability to the single-point-of-failure attacks;
- a lack of privacy for the regular people;
- an unfair inequality between different social groups;
- censorship;
- the monopoly detained by the big corporations that lets them control the terms on which the “less privileged” people and companies can interact with them;
- increased living expenses
- etc.

Centralization has arguably reached its peak by the year 2008 which saw a global financial crisis considered by many to be the worst ever since the Great Depression of the 1930s. As a response to this event, an unknown person (or a group of people) nicknamed Satoshi Nakamoto came up with the invention of Bitcoin [2] – a so-called “cryptocurrency” powered by a revolutionary piece of technology called Blockchain.

Although it was not the very first attempt at conceptualizing a chain of blocks secured through the medium of cryptography, it was the first to include in its design the extension and the management of the chain that don’t need an approval from a trusted authority.

Having been primarily used for improving the financial sector, Blockchain offered solutions to such long-standing problems as:

- double-spending; [5]
- the inability to reach an agreement without trusting the other party of the transaction (the famous Byzantine Generals' problem); [3]
- the lack of transparency;
- no privacy for the value-exchanging parties;
- high transaction costs;
- etc.

However, thanks to the universally-desired features of this technology (like its incorruptibility, consistency, integrity, persistence, decentralized consensus, security, speed and others), it is also being used to address the problems in, pretty much, every industry out there: from Education to Gaming, from Real Estate to Agriculture, from Insurance to the Internet of Things.

1.1. Motivation

Whenever our life becomes easier and more comfortable, we usually get used to its new, better form very quickly. Indeed, it's a common thing for us to forget our past difficulties, no matter how long they've been bothering us. As a rule, the solutions given to us for combating such problems are vastly underrated, especially in the long term.

Something similar has been happening to Blockchain for the past decade, ever since its invention. The majority of the relatively small number of people who have come across it, either can't see its real value and potential or simply use it, taking it for granted.

However, contrary to what one might believe, this, actually, shows that Blockchain is on the right track of development. As the technology has been getting more and more accessible for an average person, its worldwide adoption rate has been growing exponentially.

In fact, this is valid for any other ground-breaking, disrupting invention: no matter how controversial and sophisticated it is at first, if it is truly useful, it will gradually be getting widely adopted and easier to use, ultimately becoming conventional. The invention of the plow, the wheel, the refrigeration, the automobile, the Internet and the E-mail, as well as the discovery of electricity and penicillin have all shared this history.

It is hard to overestimate the importance and the actuality of the aforementioned problems. We face them with an unfortunate permanence, and they have a huge impact on our lives - whether we are aware of it or not.

Our deep concern about those problems and the desire to study the intricacies of the technology that so elegantly approaches them made us choose Blockchain as the subject for this work. Therefore, the goal of this thesis is to understand the Blockchain technology and to examine its inner structures and algorithms.

1.2. Contributions

We recall the following most important contributions of this thesis:

I. The identification and the documentation of the fundamental concepts of the Blockchain technology, as well as the relations between them.

II. The design of an architecture suitable for a Blockchain application.

III. The implementation of a resilient and stable decentralized Blockchain protocol

IV. A solution to the problem of forming the network of peer nodes supporting the Blockchain

V. Evaluation of the proposed solution

1.3. Paper Structure

Let us now present you with the structure that is followed by our thesis.

At first, it announces the problem that it is going to solve. It introduces the reader to the relevant concepts and terminology, as well as the historical point of view upon the subject. Besides, it explains some of the technical aspects of the problem and the importance of solving it for our society. Then, there's a short summary of the biggest challenges anyone working on a solution will encounter. This chapter ends with the presentation of some of the most remarkable existing solutions to the constituting parts of the problem – and it as a whole.

In what follows, the thesis goes on to listing and describing its contributions and the main components of the presented solution.

Then, there's a chapter on the tests and experiments that have been performed in order to evaluate the solution. Some tables and statistical data that help with comprehending the results of these experiments can be found here, too.

The following chapter comes with a summary of the problem on which this work is focused and the solution to it. Additionally, it features some general conclusions and ideas for enhancing the solution and for its further development.

Next, there's a bibliography containing the links to the most important materials used when working on this thesis and the Appendices section that includes anything else that is related to this work, but not directly.

2. Creating a Blockchain

In this thesis, we propose a solution to the problem of designing and implementing a decentralized Blockchain that is based on the concepts relevant to this technology.

Before delving into the issues that go hand in hand with approaching this problem, let us first define the concept of a "blockchain".

The term was inexistent at the moment of its conceptualization by Satoshi Nakamoto in 2008 [2] (at that time, the words "block" and "chain" were used separately). Nonetheless, it was perceived as a decentralized and distributed expanding chain of informational blocks that are cryptographically linked to each other.

That linkage is achieved by including the hash of a block's parent into its body, and is behind one of the pillars of the Blockchain technology: its immutability. It's impossible to alter the data inside any of the blocks without rewriting all of its successors in the chain, and doing this in such a way that the majority of the network accepts and supports it. Consequently, the older some information in the Blockchain is, the more difficult it is to alter it.

Interestingly, Blockchain can record in its history (inside the blocks, that is) any kind of information that can be represented digitally. Nevertheless, the basic unit of information inside a block is usually referred to as a "transaction", which typically constitutes an exchange of value (any sort of it) between 2 or more participants of the network.

All activity in a Blockchain network is authenticated and recorded by a distributed, global peer-to-peer network of independent nodes that collaborate with each other and adhere to a set of rules (predefined by the Blockchain protocol) in order to achieve a "decentralized consensus" [3]. This process is called "mining". Because Blockchain is fully-transparent, it's comparably easy for anyone to independently verify and audit it.

In order to come up with a solution to the announced problem, one would have to find answers to the following questions (which it directly implies):

- Who (or what) is supposed to “give birth” to the Blockchain?
- What is the way in which the network of peers supporting and using the Blockchain should be formed?
- How can the nodes reach a consensus in regards to extending and maintaining the Blockchain? [3]
- What mechanisms can the network participants use to avoid being flooded with the possible “spam” messages/requests coming from their malevolent peers trying to cause a Denial-of-Service attack? [9]
- In what way can the network nodes collectively combat the “invalid” blocks? What makes a block “valid” / “invalid”, in the first place?
- How to efficiently verify whether a piece of information (commonly referred to as a “transaction”) belongs to a block and, therefore, the Blockchain, as a whole? How should such information be organized inside a block so that it could be easily found afterwards?
- What should be done in case of a “fork” [8] (a situation in which multiple “sibling blocks” with the same height are produced by the network nodes)? What block should be considered the “righteous” successor of the top link in the chain?

It is not enough that these issues are challenging enough by themselves, but – to be truly decentralized - they must also be resolved without introducing a “trusted authority”. Along with that, the network must be robust and resilient, while its participants must be able to interact with each other with no need for trust between them.

2.1. Related Works

2.1.1 Merkle Tree

Invented by Ralph Merkle [6], this data structure can be used to efficiently verify the integrity and immutability of, pretty much, any kind of information.

Its bottom layer is constructed by splitting the information into the pieces/blocks of a certain size (specific to the function used for hashing), feeding each of those blocks as inputs to the hash function – and labeling the tree nodes with its outputs.

Any other layer of the Merkle Tree [6], unlike the bottom one, consists of the blocks that are the results of hashing the concatenation of their children that are also the output of the hash function – and not the actual bits of information.

The top element of this data structure is called the Merkle Root, and is crucial for what it offers. This single element is sufficient for verifying whether the information has diverged from its original form. The verification can be done by reconstructing the Merkle Tree of the information – and comparing the newly-obtained Merkle Root to the one that has been obtained from a trusted source.

As a matter of fact, the ability to uniquely "summarize" the data of any size into a single Merkle Root is exactly what lets blockchains be so efficient. Not being able to encapsulate multiple pieces of information into a single block in the chain, they would be truly hefty, cumbersome and way less useful.

Besides, Merkle Trees make it possible to easily confirm whether a certain node really belongs to them, knowing just a small number of other nodes (in case of a binary Merkle Tree implementation, as small as the logarithm of the total number of its bottom-level nodes).

This feature is intensively used in Blockchain for rapidly verifying whether a bit of information is a part of a block in the chain – and, therefore, the blockchain as a whole.

The alternative approach to this problem would mean a brute-force search through the whole block of information, each time such a verification needs to be done. The advantage of the

solution offered by Ralph Merkle is glowingly obvious.

2.1.2 Hashcash

The development and the increasing popularity of e-mail in the mid-1990s has indirectly produced the problem of "spamming" via this mean of communication. As a response to the latter, a system offering a solution to the just-mentioned problem was created by Adam Back in 1997 [7].

This system, named "Hashcash", limits the amount of spam e-mails being sent throughout the network by requiring the senders to expend a reasonable amount of computational power for demonstrating their good intentions. Those calculations must be done for each e-mail they want to send, and, even though this cost is negligible to the honest users, for the malevolent spammers, it means a serious drain of their resources.

The very "business model" of the e-mail spammers depends - due to the typically poor response rate for their messages - on reaching out to a huge number of recipients at a very small cost, Therefore, it makes total sense to introduce/increase the cost of sending an e-mail – and, in such a manner, cause the spammers' profit margin to dramatically shrink in size. This approach has the potential to wipe out this kind of fraud completely.

Demonstrating one's good intent within the Hashcash system implies – quite naturally – using a hash function. More precisely, a sender must find an integer "counter" which, when fed to the hash function together with the header of the e-mail being sent, produces an output with a predefined (or bigger) number of leading zeros. As of now, the most efficient method of doing that is also the most poorly performing one: the brute-force of all of the possible options.

At the same time, making sure that a sender has no ill intentions requires a single computationally-cheap, fixed-time calculation: one hashing operation. This plays a significant role in the success of the Hashcash solution [7].

Seeking to ward off malevolent senders and to reward the honest ones, many Blockchain projects have also chosen to use a Proof-of-Work strategy, having been inspired by the one proposed in Hashcash (an example of such a system will be given in the section on the Bitcoin

Blockchain). Beyond that, Proof-of-Work also helps prevent Denial-of-Service [9] attacks, which is crucial to the health of a Blockchain network.

Taking into consideration all of the above, we can state that Hashcash was one of the first Proof-of-Work systems. Besides, we believe that its creation was a major factor influencing the development of Blockchain applications – to the point at which the very existence of Blockchain (the way we know it nowadays) wouldn't have been possible without it.

2.1.3 Bitcoin

The original Blockchain proposed by Satoshi Nakamoto in 2009 [2] has been serving as a distributed transaction ledger for Bitcoin – the first and most used cryptocurrency. Thanks to it, Bitcoin became the first digital currency that had overcome the long-standing problem of double-spending the currency with no need of a trusted authority overseeing and authorizing all of the transactions in the network.

Anyone in the world is allowed to participate in the network activity: either by validating and broadcasting (in other words, "mining") new transactions and blocks, exchanging value themselves or just monitoring the network.

The mining process in Bitcoin (as well as in many other Proof-Of-Work cryptocurrencies) is based on solving the Hashcash-like [7] puzzles that require expending a certain amount of computational power.

More precisely, the process is as follows. When a miner is done collecting unconfirmed transactions that are to be encapsulated in a new block, he includes in it - alongside all of those transactions - the hash of the block that, he knows, is the latest in the network (thus, making it his block's "parent"). Then, he starts solving the "puzzle" of the block he's just constructed. His goal is for his block to generate a hash with a value that is small enough for the network to accept it. So, he starts picking the right "nonce" that, together with his block's data, will generate an appropriate hash.

The mining complexity in Bitcoin is an ever-changing variable, the value of which depends on the miners' recent success rate. It represents the minimum required number of leading

zeros in the hash of the new block.

The complexity keeps adapting so that an average time between the blocks is around 10 minutes. Every 14 days, the Bitcoin network calculates the average number of blocks per hour generated during the previous cycle, and if that number is more than 6, the complexity doubles (that is to say, the required number of zeros for the hash output of a new block grows by one); otherwise, it becomes half its current size.

It is worth mentioning that the participants' view of the network is not necessarily the same at any given time. Further still, because of its peer-to-peer nature and the data transfer latency, the network is usually partitioned into several groups that view it a bit differently. This can result in the almost-simultaneous creation of several "sibling blocks" (that are of the same "height") throughout the network. In fact, it is a relatively common phenomenon known as a "fork" [8].

Luckily, Proof-of-Work makes it possible for the network participants to always reach a "decentralized consensus" in regards to extending and maintaining the blockchain. According to the Bitcoin protocol, the only valid chain is the one that consists of the greatest number of linked blocks, starting with the genesis block. Whenever a fork [8] happens, it is up to the miners to decide which chain they want to support with their computational power (by building new blocks on top of it).

Even if the network is uniformly divided into multiple groups with different views of the blockchain, they will eventually find out about all of the diverging chains. Aside from that, it is increasingly less probable for 2 or more blocks to be created simultaneously several times in a row, which makes it easier for miners to choose the "valid" chain: the lengths of the chains ultimately grow different.

It is hard to overestimate the importance of Bitcoin [2] to Blockchain. Without it, Blockchain wouldn't even exist. However, it stands to mention that even though these technologies were inextricably linked at the beginning, Blockchain has now become something way bigger than just the "Bitcoin ledger", and should be perceived correspondingly.

3. Contributions

We recall the following most important contributions of this thesis:

I. The identification and the documentation of the fundamental concepts of the Blockchain technology, as well as the relations between them.

II. The design of an architecture suitable for a Blockchain application.

III. The implementation of a resilient and stable decentralized Blockchain protocol

IV. A solution to the problem of forming the network of peer nodes supporting the Blockchain

V. Evaluation of the proposed solution

3.1 Creating the Blockchain

To keep our solution decentralized, we have decided to embed the Genesis Block [Appendix A] of the Blockchain into the source code of our program. This made it possible for any node running the code to join the network and be sure not to be lied to by an evil-minded peer that a Blockchain he has constructed to benefit his own interests is the right one for this network.

Verifying whether someone is trying to start sending you the blocks of a wrong Blockchain is as simple as back-tracking the origin of that chain – and comparing its very first block to the Genesis Block that is bundled with the source code.

Also, having the default Genesis Block at hand allows the founding members of the network to successfully agree upon validating the first blocks following the Genesis one. Upon joining the network and realizing that no blocks have been mined yet, they consider the Genesis Block to be the only valid parent for the first block that will be created by them. After that, they start to work on creating/validating such a block.

This eliminates the possibility of having the Blockchain "fork" [8]/get divided from the very start. Unlike a fork happening later in history, such an "early" fork would generate multiple equivalent blocks, leading to the problem of choosing the "correct" block to start the Blockchain – a problem that can't be approached fairly with regards to the creators of those blocks.

Finally, it would be conceptually wrong to allow the Genesis Block to be generated like any other block, because, unlike any other future block, it doesn't have a parent – and, therefore, there can be no such field as "parent block" filled in its header. Having the Genesis Block as a ready-made, stand-alone (yet inseparable) element of the Blockchain is a simple and logical way around any possible confusion here.

3.2 Forming and Maintaining the Peer-to-Peer Network

In decentralized networks, each peer simultaneously plays the roles of a client and a server. The problem with this approach is that it is unclear how the participants of such networks are supposed to “find” each other in the first place. Moreover, one must bear in mind the possibility of the network falling to multiple independent pieces (also known as ”islands”) in case many peers lose connection to each other. On top of it, there is the challenge of programming both of the aforementioned behaviors into a single piece of code that will be downloaded and run by the nodes willing to participate in the network.

In order to avoid these ambiguities, we’ve come up with the concept of a “Genesis Peer” which would be both the core element in forming our peer-to-peer network for the first time - and the fallback option for any peer having lost touch with anyone else capable of keeping him in the network.

We’ve settled on fixing the address of the Genesis Peer to a certain value – and including it into the code of the project, so that any peer running the code would know what peer he can turn to if he’s got no one else to rely on. [Appendix B]

Aside from the Genesis Peer, a new participant can also be introduced to other peers by a “friend” of his who is already a part of the network and whose address he initially knows.

It’s important to mention that even though this approach might seem to be an element of centralization, it is, in fact, very different from it. First of all, it’s just the address of the Genesis Peer that is pre-established, not the very peer, and so, anyone can play its role. Secondly, unlike the central authority from any centralized solution, the Genesis Peer doesn’t get any super powers in the network, but abides by the very same rules that all the other peers comply with.

Another safety measure that helps keep the network from falling apart is that a certain number of peers with whom a node communicates is being stored persistently [Appendix C]. If it so happens that the node goes offline, they are the very first peers that it’ll try to connect to once it’s back online. [Appendix D]

3.3 Reaching Consensus: Validating Blocks, Extending Blockchain, Forking

Unlike a centralized system, a Blockchain application needs a fair, efficient and robust mechanism for all of its users to be able to reach a decentralized consensus in regards to the state of the Blockchain in real-time and at any moment of its existence.

In order to provide such a mechanism, we've created a set of clear and strict rules that are to be followed by the network participants. These rules are fused into the code of the project, and, therefore, are known to the nodes running them from the very beginning.

Combatting invalid blocks is just one issue that such rules help all of the network nodes to solve. Each of the nodes is aware that a block is considered "valid" only if it satisfies the following criteria [Appendix E]:

- ✓ it is of the same type as the one of the Genesis Block;
- ✓ the hash of its "parent" (found in its header) is the same as the hash of the latest block in the Blockchain;
- ✓ the Merkle Root [6] of its contents is identical to the one specified in its header;
- ✓ the value of its header hash is smaller than the Proof-of-Work target of the Blockchain.

So, when a node detects that it has been sent an invalid block, it not only refuses to add the block in its version of the Blockchain, but, also, doesn't forward this block to its other peers. In other words, the broadcast of an invalid blocks in the network is halted by the first honest node that intercepts it.

The latter, by the way, is crucial to the collective combat of a Denial-of-Service [9] attack based on trying to flood the network with numerous blocks, forcing it to choke on all those blocks and rendering it useless. Such an attack would, obviously, be performed using invalid blocks that are very cheap to make (for instance, a block can be copied from the Blockchain – and re-transmitted all over again, multiple times). Such blocks would, then, be transmitted in vast numbers to as many network participants as possible.

The aforementioned set of rules is known to each individual in the network, and so, everyone can validate new blocks independently. Therefore, such an attack would be close to a waste of time for the attacker.

However, rejecting invalid blocks arriving in big numbers can be relatively time-consuming. Therefore, we've decided to validate each new block in a separate thread that executes independently and, consequently, can't cause a slowdown of the thread where the communication with the block sender happens.

At the same time, the valid blocks get propagated throughout the network very fast,

because every node that plays by the rules instantly shares the news about the new valid blocks with his peers, who, in their turn, spread it further. [Appendix F]

This is the way an honest node reacts to a new valid block even while it is mining a block of its own. According to the rules, its half-mined block won't be accepted by the other peers when the valid block it has just received will be spread out in the network (as these blocks would unequally compete for the same spot in the Blockchain). In order not to waste any more computing power producing a block that is doomed to be rejected, it halts the current mining process – and starts a new one that'll be building a child for the newly-accepted block. [Appendix G]

Having said that, we've arrived at another possible roadblock: the situation that happens when multiple blocks of the same "height" (competing for the same place in the chain) are created and propagated through the network. If there were no predefined rules, this would inevitably lead to the fragmentation of the network. However, with those rules in place, each peer knows the following:

- at any time, there can be only one valid Blockchain: the one with the most cumulative Proof-of-Work (in the form of blocks) in it;
- the dilemma of a fork [8] is resolved by the aggregating power of each node's individual decision as to which of the new sibling-blocks to support;
- the block that is recognized by the majority as the rightful successor of the block chain becomes just it.

So, when a node must decide which of the many sibling blocks to support, it compares the numerical representation of their Proof-of-Work – and chooses to support the one with the biggest value. Taking it one step forward, if the Proof-of-Work of all of these blocks happens to be the same (the probability of which is extremely small), the "right" block is determined based on the value of the hash of each of their headers. The one with the smallest header hash (which, probabilistically-speaking, was the hardest to generate) is chosen. [Appendix H]

All of the above makes it possible for all of the network participants to reach a decentralized and trustless consensus in regards to the state of the Blockchain at any given time.

4. Evaluating the Proposed Solution

When it comes to examining our solution, we've performed a series of tests aimed at evaluating the following core functionalities and features of the system:

- Functionality;
- Reliability;
- Stability;
- Efficiency.

Let us present you with what exactly we've tested – and how we did it.

4.1 Functionality

First things first, we've tested the ways in which our program:

- uses the Elliptic Curve Digital Signature Algorithm when generating the private and public keys for a node; [Appendix I]
- computes the Merkle Roots [6] of blocks of information; [Appendix J]
- communicates with other nodes;
- computes hashes and parses the hashes computed by others;
- validates the block structure;
- validates the block miner's Proof of Work;
- reacts when receiving a valid/invalid block;
- behaves when a fork [8] occurs. [Appendix H]

See *Table 1* for a head-to-head comparison of our Blockchain with the Bitcoin Blockchain.

Table 1: Comparison between our Blockchain and the Bitcoin Blockchain

Feature	*Presence in the Bitcoin Blockchain*	*Presence in our Blockchain*
Elliptic Curve Cryptography	✓	✓
Merkle Root integration	✓	✓
Distributivity	✓	✓
Decentralization	✓	✓
Mining based on the Proof-of-Work	✓	✓
Resistance to invalid blocks spamming	✓	✓
Resistance to the chain forks	✓	✓
Well-defined block structure	✓	✓
Extensibility of the P2P network	✓	✓
Maintaining a well-balanced connectivity of a node	✓	✓
High Byzantine fault tolerance	✓	✓
Decentralized consensus	✓	✓
Smart Contracts support	✓	✗
Lightweight clients support (Simple Payment Verification)	✓	✗
Enhanced scalability (Lightning network)	✓	✗

4.2 Robustness

Secondly, we were interested in how well the system performs in and recovers from the problematic/„abnormal" situations. In other words, we wanted to make sure the program behaves when things don't go exactly as planned.

So, one of the things we did was pushing the peers (25 of them) into the network all at once (as much as it is possible in a manual test), instead of inter-connecting them at a slower

pace (which had already been tested before). At the same time, we were connecting the nodes to both the default peer and certain peers in the network.

Another thing that got our attention was the reaction of the nodes to an active peer suddenly dropping out of the network – and the recovery of that very peer from such an event upon his re-connection to the network. So, we caused multiple chaotic disruptions in the network connections (mainly, by killing the processes keeping the nodes alive), followed by the equally chaotic "renaissance" of the disconnected nodes.

What we observed was that, thanks to the algorithms we've created for treating similar situations, the nodes that were having their peers disconnected did not start to fall apart, but, rather, gave their disconnected peers several more attempts to go back online, and, when/if that didn't happen, forgot about them, trying to replace them with some other active peers from the network. [Appendix K]

On the other hand, the nodes that re-joined the network tried, at first, to re-connect to the peers they used to know (getting this information from the persistence) [Appendix D] and, for every past peer that became unavailable, tried to connect to some other active nodes in the network.

4.3 Stability

Our next concern was the stability of the system. Especially in the longer run and when it is being „stressed".

One thing we did to test this was to leave the network of inter-connected nodes running for a long time (see *Table 2*). What we noticed by the end of that period was that the blockchain grew to having thousands (4k+) of blocks in it, and was being extended and maintained by the network of nodes as before. All the while, the views of the different nodes upon the state of the blockchain have not diverged, even though dozens - if not hundreds - of forks [8] have occurred during those hours. Generally speaking, everything kept functioning normally.

Table 2: Monitoring the network behavior

Amount of time elapsed	Approximate number of blocks mined in the network	Number of issues spotted
5 min	75	0
10 min	150	0
30 min	450	0
1h	900	0
2h	1,800	0
5h	4,500	0
10h	9,000	0
12h	**10,800**	**0**

The other experiment involved freezing (for hours on end) the processes keeping the nodes active – and, then, suddenly, re-starting them. Neither has this led the system into a deadlock or broke it in any way.

Last but not least was the decrease of the block mining complexity which led to the increased load upon the system, as the new blocks were appearing more often throughout all of the network – and, therefore, the nodes had way more information to parse, evaluate and react to. Aside from this, the probability of one (or more) forks happening at any moment had also soared, which represented an even bigger challenge. Nevertheless, no remarkable issues had been provoked by this experiment.

One of the main reasons for the stability of our program is the fact that there are separate threads responsible for:

- monitoring the state of the blockchain – and spreading the word about its new (or replaced) blocks throughout the network; [Appendix F]
- monitoring the peer connections and if/when it so happens that one of the respective peers gets unresponsive, making sure that the node is connected to as many known peers from the network as possible; [Appendix M]

4.4 Efficiency

At all times during the development of our program, the efficiency of the end-result

program has been a priority for us. Therefore, it was a big surprise to uncover a serious problem in one of the working threads of our application.

We spotted the issue during an experiment involving monitoring the system when adding more and more nodes to the network. The thread that monitors the state of the blockchain and notifies all of the active peers of the node [Appendix F] was very inefficient about using the CPU time – and caused some sizeable delays in the network communication (which is crucial for this kind of programs). Thanks to that experiment, we have not only implemented a more resource-efficient algorithm for broadcasting the state of the blockchain to the other peers (see *Table 3* for more details), but have also prevented any side-issues caused by the opposite of that.

Table 3: CPU load before and after the introduced optimization

Point in time	***Average CPU load***
Before the optimization	90-100%
After the optimization	5-10%

To sum it all up, the results of the conducted experiments have led us to believe that we've created a program that performs well according to the proposed standards.

5. Conclusions

Within the context of this thesis, we've tackled the problem of coming up with an appropriate design for a decentralized Blockchain program and, also, an implementation for it.

The end-result of our efforts is, arguably, a stable, fault-tolerant, correct and scalable Blockchain program that comes with mechanisms for reaching a trustless, decentralized consensus, combatting invalid blocks and spam, dealing with forks [8] and many more.

In terms of future works on this project, we'd very much like to see it being built upon when working on some other problem. Except the obvious processing of payments/value transfers, it could, also, be used to effectively monitor the supply chain of a business, correctly and fully assure the respect of the copyright upon some creation, justly transfer and maintain the rights of a lawful owner of some property and so much more… The list of potential applications for this technology could really be continued indefinitely.

Now, concerning the technical side of the project, what could be improved is the "user-friendliness" of our program. While sufficient for the scope of our thesis, the user interface could be made more abstract from the underlying technology, so that the user doesn't have to know and be exposed to so many technical details of the program he's interacting with.

Another aspect that could be enhanced is the way in which a node manages connections to its peers. Having in mind the possibility of malevolent actors participating in the network, we've decided to impose restrictions upon the number of active incoming and outcoming connections maintained by a node. Besides, we've introduced high standards that anyone willing to be trusted by his active peers and not to lose connections to them must satisfy. However, this protective mechanism definitely has some room for improvement, as any limit of allowed connections will become a bottleneck for the network of a certain size. An idea would be to change the limit dynamically with respect to the number of participants in the network.

One more thing that comes to mind when thinking about the potential improvements is the algorithm we've created for resolving the "fork" [8] situations. At the moment, there are several layers of guards that protect the network from the risk of being divided every time a fork happens. We've designed the system to resolve each fork as soon as it appears – and move on. At

the same time, some more sophisticated solutions (like the Bitcoin Blockchain [2]) allow a way bigger number and complexity of forks – even recursive/exponential forks. At any point in time, such systems analyze (retroactively) all of the existing chain lengths (produced by forks, various in their impact and age) – and deduct the single "valid", main chain which is always the longest one.

6. Bibliography

- [1] Andreas Antonopoulos: *Mastering Bitcoin - Programming the Open Blockchain*, 2014 (https://github.com/bitcoinbook/bitcoinbook)
- [2] Satoshi Nakamoto: *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008 (https://bitcoin.org/bitcoin.pdf)
- [3] Leslie Lamport, Robert Shostak, Marshall Pease: *The Byzantine Generals Problem*, 1982 (https://people.eecs.berkeley.edu/~luca/cs174/byzantine.pdf)
- [4] Iaroslav Mazur: *Creating a Blockchain*, 2020, the source code (https://github.com/Iaroslav-Mazur/License-Paper)
- [5] Mark Ryan: *Digital Cash*, School of Computer Science, University of Birmingham (https://www.cs.bham.ac.uk/~mdr/teaching/modules06/netsec/lectures/DigitalCash.html)
- [6] Ralph Merkle: *Method of providing digital signatures*, 1979 (https://patents.google.com/patent/US4309569)
- [7] Adam Back: *Hashcash - A Denial of Service Counter-Measure*, 2002 (http://www.hashcash.org/papers/hashcash.pdf)
- [8] Till Neudecker, Hannes Hartenstein: *An Empirical Analysis of Blockchain Forks in Bitcoin* (http://ifca.ai/fc19/preproceedings/24-preproceedings.pdf)
- [9] USA National Cyber Awareness System: *Understanding Denial-of-Service Attacks*, 2004 (https://www.us-cert.gov/ncas/tips/ST04-015)

7. Appendices

Appendix A: Adding the Genesis Block to the Blockchain

```
with open('genesis_block', 'rb') as f:
      blockchain.append(pickle.load(f))
```

Appendix B: Trying to connect to the Genesis Peer first

```
if first_peer_port_nr != None and \
      (first_peer_port_nr != GENESIS_PEER_ADDRESS or my_port_nr !=
GENESIS_PEER_ADDRESS):

      peers_socks_vers_out_lock.acquire()

      connection_result = connect_to_a_peer(first_peer_port_nr)
      if type(connection_result) is tuple:
            peers_socks_vers_out.append(connection_result)
            …
      peers_socks_vers_out_lock.release()

      if type(connection_result) is tuple:
            socket_to_first_peer = connection_result[0]
            connect_to_more_peers(get_peer_ports(socket_to_first_peer))

      elif type(connection_result) is list:
            connect_to_more_peers(connection_result)
```

Appendix C: Storing the addresses of the active peers in the persistence

```
def remember_peers():
      file_name = "recent_successful_connections" + str(my_port_nr)
      with open(file_name, "w") as f:
            for port in active_peers_ports:
                  f.write(str(port) + '\n')
```

Appendix D: Trying to reconnect to the stored peers first

```
file_name = "recent_successful_connections" + str(my_port_nr)
      if os.path.isfile(file_name):
            with open(file_name,'r') as f:
                  for line in f.readlines():
                        recent_peers.append(int(line))
            if len(recent_peers) > 0:
                  connect_to_more_peers(recent_peers)
```

Appendix E: The set of rules for combatting the invalid blocks

```
def is_block_valid(block, parent_block):
      if type(block) != type(blockchain[0]):
            …
            return False

      parent_block_hash = parent_block.get_hash_hex()
      if block.block_header.prev_block_hash != parent_block_hash:
            …
            return False

      candidate = tx.Candidate_Block(block.transactions)
      tx_merkle_root = candidate.get_merkle_root()
      if block.block_header.merkle_root != tx_merkle_root:
            …
            return False

      if block.block_header.pow_target < POW_TARGET:
            …
            return False

      header_hash = block.get_hash_hex()
      if int(header_hash, 16) >= POW_TARGET:
            …
            return False

      return True
```

Appendix F: Broadcasting a new valid block in the chain

```
while True:
```

```
blockchain_len = len(blockchain)
new_highest_block_hash = blockchain[blockchain_len-1].get_hash_hex()

blocks_to_broadcast = []
[determining what blocks are to be broadcast]

highest_block_hash = new_highest_block_hash

for block_nr in blocks_to_broadcast:
      block = blockchain[block_nr]
      block_tuple_pickled = pickle.dumps((block_nr, block))

      peers_socks_vers_out_lock.acquire()
      [don't broadcast the block if it's not in the chain anymore]

      for sock_ver in peers_socks_vers_out:
            [sending the block to all of the connected peers]

      peers_socks_vers_out_lock.release()
Time.sleep(1)
```

Appendix G: Halting the mining process if a new valid block has been received

```
chain_height = len(blockchain)
if chain_height > block_to_be_mined_nr:
      …
      return wait_long_enough(chain_height)
```

Appendix H: Guards against the forks

```
if new_block_id == len_blockchain:
      #accept the direct child of the highest block
      parent_block = blockchain[len_blockchain-1]
      if miner.is_block_valid(new_block, parent_block):
            blockchain.append(new_block)
            …

elif new_block_id > 0 and new_block_id == len_blockchain - 1:
      #accept new block with the same height as the highest one
      parent_block = blockchain[new_block_id-1]
      highest_block = blockchain[new_block_id]

      if miner.is_block_valid(new_block, parent_block):
            new_block_hash = new_block.get_hash_hex()
            highest_block_hash = highest_block.get_hash_hex()

            # accept only if it has a bigger Proof Of Work:
            new_block_nonce = new_block.get_nonce()
            highest_block_nonce = highest_block.get_nonce()
            if new_block_nonce > highest_block_nonce:
                  blockchain[new_block_id] = new_block
                  …

            elif new_block_nonce == highest_block_nonce:
                  if int(new_block_hash, 16) < int(highest_block_hash, 16):
                        blockchain[new_block_id] = new_block
                        …
```

Appendix I: Using ECDSA for generating the public and private keys

```
def generate_a_private_key():
      random_bytes = os.urandom(32)
      private_key = int(binascii.hexlify(random_bytes),16)
      return private_key

def generate_the_public_key_point(private_key):
      pub_key_point = private_key * generator
      return pub_key_point

def compress_the_public_key_point(pub_key_point):
      compressed_result = ''
      if pub_key_point.y() & 1:
            compressed_result = '03' + '%064x' % pub_key_point.x()
      else:
            compressed_result = '02' + '%064x' % pub_key_point.x()
      return compressed_result
```

Appendix J: Calculating the Merkle Root

```
merkle_tree = []
for tx in self.transactions:
      tx_hash = [the tx data encoded twice via sha256]
      merkle_tree.append(tx_hash)

if len(merkle_tree) % 2 == 1:
      merkle_tree.append(merkle_tree[-1])

while len(merkle_tree) > 1:
      merkle_tree_new = []

      for tx_hash_nr in range(0, len(merkle_tree)-1, 2):
            pair = merkle_tree[tx_hash_nr] + merkle_tree[tx_hash_nr+1]
            pair_hash = [the pair data encoded twice via sha256]
            merkle_tree_new.append(pair_hash)
      merkle_tree = merkle_tree_new
return merkle_tree[0]
```

Appendix K: Giving the disconnected nodes several chances to come back online

```
if peers_port in failed_still_alives:
      failed_still_alives[peers_port] += 1
      if failed_still_alives[peers_port] > 5:
            …
            forget_a_peer(sock_ver)
else:
      failed_still_alives[peers_port] = 1
```

Appendix M: Monitoring the peer connections

```
nr_outer_conns = len(peers_socks_vers_out)
if nr_outer_conns < MAX_NR_OF_CONN_OUT and STATE_CATCHING_UP == False:
      curr_peers_number = len(active_peers_ports)
      [if enough time has passed or the number of active peers has changed]
            latest_maximization = Time.time()
            peers_number = curr_peers_number

            for i in sample(range(nr_outer_conns), nr_outer_conns):
                  socket = peers_socks_vers_out[i][0]
                  connect_to_more_peers(get_peer_ports(socket))
```