# Micropayments through cryptocurrency mining

Department of Informatics and Media

UPPSALA UNIVERSITET

Author: Viktor Tigerström
Supervised by: Andreas Hamfelt

September, 2016

# Abstract

The monetary policies of states and systems built upon them do not naturally allow transactions of a very small value, as the transaction costs exceeds the actual value of the transaction. These types of transactions are called micropayments. This is problematic as it removes the possibility to monetize content that has a valuation that is so low that the costs of the transaction exceeds the value of the content.

In this thesis we aim to create a system that allows micropayments to monetize low value content. We do so by developing a design theory based on Gregor and Jones conceptual model for design theories within Information Systems research. The system that we develop will use the end users computational power to generate a value, by running a cryptocurrency miner.

We present the background knowledge required to fully understand the presented design theory. Within the design theory, we present a theoretical framework to base systems on that enables micropayments through cryptocurrency mining. We also present a developed proof of work prototype that proves the validity of the theoretical framework.

Lastly we discuss our design theory. We conclude that the design theory enables transactions of a very low value, such as 0,0001 $ cents. Transactions of such small value is not possible with systems built upon states monetary policies. We also conclude that the proposed design theory can be further developed to function independently of cryptocurrency mining. Instead the value for the transaction could be generated through solving complicated problems if institutions are willing to pay for computational power to solve them.

# Contents

# Chapter 1

# Introduction

## 1.1 Introduction

Systems created upon monetary policies of states to allow digital payments do not naturally allow transactions of a very small value. These type of transactions are called micropayments. The reason for this is that the transaction cost of each individual micropayment, is significant and sometimes exceeds the actual value of the transaction.

Imagine a blogger that has an average reader base of 1 million unique readers per day. If the blogger wants to start profiting from the content the blogger is writing, the blogger has three main income alternatives. Either make the blog available for monthly paying subscribers only, or include advertisements, or lastly accept donations. The first two alternatives will likely lead to a decrease in readers. The last alternative is optional, and will therefore not lead to a decrease in readers. However only a small percentage of all readers will make a donation, as the donation will have to be of a significant value to exceed the transaction costs. If the readers instead would be able to for example donate only 1 USD cent, each donation will not be of any significant value. However given that 5% of all readers chose to donate 1 USD cent, the collective value per month of all donations reaches 15,000 USD.

With the current banking systems, this low value in a transaction is simply not possible. In this thesis we propose an alternative system that would allow micropayments with this kind of value. This will be done by instead using the computational power of each individual client (in this case reader), to generate a value corresponding to the value of the micropayment.

The solution that we propose is a theoretical framework, which to base systems upon to allow micropayments of this kind of value. To prove the function of the theoretical framework a functioning prototype has been developed to prove its validity.

## 1.2 Micropayment definition

There is no absolute definition of how small the transaction value must be to be called a micropayment. It is therefore subjective. In this thesis we define that a transaction is a micropayment if it is valued at 1 USD cent or less.

## 1.3 Method

In this thesis we develop a design theory, based on Gregor and Jones contribution. Gregor and Jones developed a theoretical framework that describes a conceptual model of how a design theory within Information Systems research should be developed. The model consists of 8 core components:

1. *Purpose and scope*: The component describes a set of meta-requirements or goals for the artifact and defines the boundaries or scope of the theory. In short it defines "what the system is for".
2. *Constructs*: The entities of the theory and how they are represented.
3. *Principle of form and function*: The architecture of the artifact.
4. *Artifact mutability*: The artifact's changes of state anticipated in the theory.
5. *Testable propositions*: Statements about the design theory that can be tested as true or false.

6. *Justificatory knowledge*: Defines the knowledge based on social or design sciences that justifies the design theory in the context.
7. *Principles of implementation*: Describes the principles that was used when implementing the artifact in the design theory.
8. *Expository instantiation*: The physical implementation of the artifact represents the design theory.

All of these components must be evaluated when creating a design theory. The design theory that we have developed within this thesis will be evaluated analytically. (Shirley Gregor 2007)

## 1.4 Aim

The aim of this thesis is to develop a design theory based on the contribution of Gregor and Jones. The design theory describes how cryptocurrency mining can be used to make micropayments. We start off by first providing necessary background information for understanding how cryptocurrency and blockchain technology works, and how value can be generated through cryptocurrency mining. After this a theoretical framework of how cryptocurrency mining can be used to make micropayments is provided. Lastly we provide a description of a developed functioning prototype, an artifact, that proves that the theoretical framework is valid.

## 1.5 Research question

This thesis is centered around creating a solution that enables micropayments through cryptocurrency mining. The design theory created in this thesis, has been formed by a research question. This research question is:

Is it possible to create a solution that enables micropayments, through the use of cryptocurrency mining?

## 1.6 Outline

This thesis first provides background information regarding cryptocurrency and blockchain technology. The background information is generally focused on the cryptocurrency bitcoin when giving detailed explanation. It has intentionally been focused on areas that can be generalized to most other types of cryptocurrencies. The information regarding cryptocurrency and blockchain technology provided in the background section has been selectively chosen to understand the design theory that this thesis proposes.

The next chapter describes a theoretical framework of how cryptocurrency mining can be used at the to make micropayments. The provided framework defines guideline recommended to follow when creating a system that allows micropayments through cryptocurrency mining. The framework is generalizable enough to allow multiple ways of implementing the system.

The following chapter describes a specific way of implementing the theoretical framework, by providing a description of a functional prototype to prove the proof of concept of the theoretical framework. The prototype is just a proof of concept and is therefore not fully operational for a full scale business system. Because of this we also provide information regarding how the prototype can be further developed to be able to operate as a full scale business system.

After this we provide a discussion to conclude that the thesis fills the requirements of Gregor and Jones' theoretical framework of a design theory. We also conclude the developed prototype as a proof of concept. Further more, we conclude that the prototype would function in a full scale system if developed further.

Lastly potential future research is provided.

The 8 core components of Gregor and Jones contribution is approached in the different chapters of this thesis. The *Purpose and scope* is stated in chapter 1. Chapter 2 provides *Justificatory knowledge*. The components *Principle of form and function*, *Constructs*, *Artifact mutability*, *Testable propositions* as well as *Principles of implementation* are described in chapter 3. Lastly the component *Expository instantiation* is stated in chapter 4.

# Chapter 2

# Background

## 2.1 History of digital currency

The history of digital currency is closely linked to the developments of cryptography. The reason for this is that in order to have a secure and reliable digital currency, there are fundamental properties linked to cryptography that have to be achieved:

*Trust in the authenticity of the currency*: In order for a digital currency to be viable, a user of the currency must be able to tell if the digital currency is authentic, or if it is counterfeit.

*Possibility to claim the tokens of the currency*: A user of the currency must be able to ensure that they are the only ones that can spend the tokens of the currency that they hold. The user must also not be able to spend the same tokens more than once. This is problematic as the currency is digital and anything digital can be copied (also known as the double spending problem).(Chapter 1 Antonopoulos 2014)

As two of the main goals of cryptography is *authentication* and *non-repudiation* the close link between digital currency and cryptography is explainable. Many different solutions to develop digital currencies have been tried throughout history. (page 5 Menezes, Oorschot, and Vanstone 1996) Before 2008, there was however never any solution that managed to solve the double spending problem in a viable way. Since the currency is digital, there has to be some kind of solution to make sure that users can not just copy their digital tokens and spend the same tokens more than once.

Before 2008 the only solution found to this problem was to have a central authority that controls all the transactions. When a transfer of a digital token has taken place, this central authority updates which user has the ownership of the digital token. This is to make sure that the new owner of the digital token is the only user that can later claim ownership of the digital token and transfer it. This way of solving the problem, is similar to the way the banking systems presently works. A problem with this solution, is that governments has the monetary policy of being the only authority within countries that has the right to mint currencies. This is problematic, since when an organisation or person tries to mint a currency, governments or other organisations may try to shutdown the digital currency.To do so, all that needs to be done is to shutdown the central authority that controls the transactions of the digital currency. (Chapter 1 Antonopoulos 2014) Two examples of digital currencies invented before 2008 are b-money and HashCash.

## 2.2 Cryptography

Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone defines cryptography as the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and origin authentication. (page 4 Menezes, Oorschot, and Vanstone 1996)

Cryptography is a set of techniques rather than only a means of providing information security. This implies that cryptography focuses on the prevention and detection of cheating and other malicious activities. (page 4 Menezes, Oorschot, and Vanstone 1996)

Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone defines four goals of cryptography, upon which all further specific goals within cryptography derive from. These are:

1. *Privacy or confidentiality*: One of the main goals of cryptography is to keep the content of information private or confidential only to those that are authorized to take part of it. There are numerous different

cryptographical techniques to achieve this goal, and which technique is most suitable depends on the specific case. One example would be a scenario where the contents of an email should be readable only by the receiver, which for example could be achieved by encrypting the email text using a key that is known by the receiver. (page 5 Menezes, Oorschot, and Vanstone 1996)

2. *Data integrity*: The goal of data integrity is to ensure that data cannot be altered by an unauthorised part. To solve this issue techniques to detect data alteration by unauthorized parties have been developed. (page 5 Menezes, Oorschot, and Vanstone 1996)

3. *Authentication*: The goal authentication relates to techniques to authenticate both information and the entities sharing it. For entity authentication this implies that the entities sharing the information should be able to authenticate that they are the exact entity they claim to be. Information authentication techniques guarantee that the origin of the information actually is from the actual entity. Because of these two focuses of authentication, the goal of authentication has been separated into two major classes: entity authentication and data origin authentication. (page 5 Menezes, Oorschot, and Vanstone 1996)

4. *Non-repudiation*: The goal of non-repudiation is to prevent an entity from denying previous commitments or actions. This means that an action that has been made by an entity is final and should never be possible to deny or undo. (page 5 Menezes, Oorschot, and Vanstone 1996)

## 2.3 Bitcoin

In the original white paper that pseudonym Satoshi Nakamoto published in 2008, a concept of a digital decentralized cryptocurrency called bitcoin was proposed. The cryptocurrency was proposed to be an application hosted on an invention called a blockchain. Satoshi Nakamoto combined previous ideas of creating a digital currency, such as b-money and HashCash. By mixing these ideas with known cryptographic concepts Satoshi Nakamoto found a solution of how to transfer digital currency between different parts on a decentralized network without the need of a central authority. Using the concept of a blockchain, these transactions of the digital currency could guarantee that the transferred tokens were not counterfeit. (Nakamoto 2008)

## 2.4 Blockchain

The word blockchain refers to a technology that was proposed by the pseudonym Satoshi Nakamoto through the public release of the "The bitcoin white paper" in 2008. Satoshi Nakamoto suggested a cryptographic solution to take a hash of a block of items. By timestamping a block and then publishing the hash to every participant in a public network the timestamp will prove that the data in the block existed at the time of the timestamp. On top of this, every new block of items that is later published in the network, will also include the previous timestamp in its hash. This leads to a chain of hashes of blocks of items, hence the word blockchain. This implies that the validity is reinforced of every block published prior to the new block. (Nakamoto 2008)

In other terms a blockchain is a public ledger holding transactions that is distributed over a decentralized network. Every participant in the network holds a copy of the ledger, and is also informed about any new transactions that are added to the ledger. This leads to that every item that is transacted in the ledger can be traced back to its origin. This is done by following all transactions it has been part of back to its creation and original inclusion in the ledger. This means that as long as the users of the ledger are unable to add items to the ledger, it is impossible for the users to counterfeit an item in a transaction. The reason for this is that the ownership of the item can be validated by tracing its origins in the ledger.

The solution that Satoshi Nakamoto proposed, also guarantees that a user A transferring an amount of digital tokens to a user B, can not transfer the exact same tokens to another user C after the transaction to user B. This is also known as not allowing double spending. (Nakamoto 2008)

## 2.5 The double spending problem

Before the release of The bitcoin white paper, there have been several attempts of creating digital currency. All had however failed to provide a viable solution to the double spending problem without using a central clearing authority. A simple explanation of the problem is that all digital currencies in their core is just a set of digital data. Like any set of digital data, it can easily be copied. If a user A has digital tokens of a digital currency that it controls on their digital storage media, this means that the user A can transfer to the user B's digital storage media. The problem is that the user A can easily copy the data before transferring it, and then later claim ownership of the data and transfer the same data to a user C, hence spending the digital tokens more than once. (Chapter 1 Antonopoulos 2014)

To solve this issue, Satoshi Nakamoto proposed a new solution in The bitcoin white paper. The solution did not have a central authority that controls all transactions. The users did not have actual data representing digital tokens that was transferred between the users either. Instead the transfers took place through a public ledger in a decentralized network that was distributed to all participants of the network, the blockchain. It is then easy to determine if a transaction is valid and does not lead to a double spending. This is done by checking if the items transacted have already been spent on the ledger, as it is public. This does however require a guarantee that the ledger holds correct information. (Nakamoto 2008)

## 2.6 Byzantine generals problem

The bitcoin white paper includes a solution that describes how to guarantee that the participants of the network can know that the information in the ledger is correct. It describes how computers without the need of a third party can agree on a state, for example that user A has made a transaction to user B. This problem is know as the Byzantine Generals problem, and is a know problem within computer science. (Leslie Lamport and Pease 1982) Satoshi Nakamoto proposal was the first solution to solve the problem. The core of the solution was to make the participating computers in the network to compete against each other to solve a cryptographic problem. On top of this, in order for a participant to get its solution accepted by the rest of the network, the participant have to follow a set of rules that every participant of the network also have to follow. By combining these two concepts, the validity of the decentralized ledger can be guaranteed. This process has been given the name mining. (Chapter 1 Antonopoulos 2014)

## 2.7 Mining

The main function of the mining process is to secure the network in order to guarantee the validity of the blockchain. However the process has been given the name mining based on that mining generates new minted cryptocurrency. When a new block to the blockchain is published, the participant that has found the cryptographic solution and publishes the block is rewarded with an amount of bitcoins. The amount of bitcoins that is rewarded per block is not permanent and is decreasing over time as more blocks are added to the blockchain. Therefore the way that bitcoins are generated can be compared to the mining of gold, where the gold gets extensively harder and harder to find over time, hence the name mining. (Chapter 8 Antonopoulos 2014)

In its simplest terms mining can be seen as a cryptographic game where every participant in the game randomly suggests different solutions that when run through an encryption algorithm proves to be correct or not. It is impossible to determine if a solution is correct or not before it is run through the encryption algorithm. Therefore every participant has no choice but to suggest random solutions as it is impossible to determine which solution will be correct before trying it. This means that the game is constructed to make it difficult to find a correct solution, but very easy to check whether a solution is correct or not. (Chapter 8 Antonopoulos 2014)

Based on these characteristics a metaphor to compare the mining process to a competition to solve a giant sudoku puzzle is often used. This is since it is hard to find the solution for a sudoku puzzle, but easy to check if the solution is correct or not.

## 2.8 Incentives for miners

An important aspect of the concept of mining is that every participant is investing resources that have value in order to participate in the mining competition. These resources are mainly electricity and hardware. This means that the miners are economically invested in the mining process, and therefore have incentives for getting return on their investment. In order to get return on their investment, the miners need the reward that they get from finding the correct hash for a block. The miners also need to make sure that the block is constructed in a correct way to get it accepted into the blockchain. If the found block is not accepted into the blockchain, the miner that found it does not get the reward it includes. In order to get a block to be accepted, the miner needs to make sure that the block follows all the rules set up by the consensus of the network. If a miner starts changing parameters in the block, for example trying to change the reward size of bitcoins, the other miners of the network will not accept the new block and the block will therefore not become a part of the agreed blockchain. (Nicolas T. Courtois and Naik 2014)

This is essential as this gives the miners incentives to follow the rules if they want to maximize their profits. This also means that the miners benefit from making sure that the rules are being followed by others. The miners do this by not sending blocks that are invalid further to other miners in the network, and by accepting valid blocks and send them further. It is possible to imagine a scenario where a miner did the opposite. The actions of one miner would not make any difference, as one miner itself cannot create a consensus on the network unless this miner controls over 50% of the hashing power. However if enough miners to create a consensus did let invalid blocks through, or stopped transmitting valid blocks further, this would seriously damage the network as the blockchain could not be seen as reliable. This would in turn negatively affect the value of bitcoin as it would become less useful. As the miners have invested resources to mine, this would be very unbeneficial for the miners. The reason for this is that the value of the reward per block, is dependent on the value of bitcoin. (Nicolas T. Courtois and Naik 2014)

This leads to the conclusion that one of the key aspects to why consensus can be found among the miners is that the reward for following the rules is much higher than breaking them. This gives the miners an incentive to follow the rules.

## 2.9 Blockchain related cryptography

Some cryptographical techniques are more central within blockchain technologies. This section aims to give a basic understanding of these specific techniques.

1. *Encryption*: Encryption refers to techniques to scramble information into an unrecognisable form. This scrambled unrecognisable form is called a hash. The idea of encryption is that it should not be possible to restore the hash to its original form if a part only has access to the scrambled information. A encryption technique is defined as breakable if a third part with no prior knowledge of the encryption scheme the technique uses, can restore the scrambled information to its original form within a appropriate time frame. (page 14 Menezes, Oorschot, and Vanstone 1996)

   Some encryption techniques allows the authorized parts to decrypt the information to its original form with the use of a key that is only known by trusted parties. A key is a combination of different characters. There are two main forms of key techniques, secret key techniques and public key techniques. Secret key techniques are based on the idea that there is a secret key that that only authorized parts know and that can decrypt the scrambled information. Public key techniques are explained below. (page 15-21 Menezes, Oorschot, and Vanstone 1996)

   Blockchain technologies use encryption for many different areas. One clear example of this is during the the mining process. Blockchain technologies use an encryption function at the block header in order to see if the resulting hash meets the difficulty target. The hash is irreversible which makes it impossible to know which variation of a block header will meet the difficulty target before applying the encryption function to it. Bitcoin encrypts the block header by applying the encryption function SHA-256 twice to the block header.

2. *Public key techniques*: The idea of public key techniques is that there

are two separate keys involved in the encryption and decryption process of a set of information. There is one private key, that is secret to everyone except the part that encrypts the information. There is also one non secret public key that is publicly known. The public key is generatable from the private key, but it is not possible to generate the private key based on the public key. The idea is that information that has been encrypted with the private key, can be decrypted using the public key. In this way it is possible to guarantee that the part that has encrypted the information is the authorized owner of the private key. This is done by checking if it can be decrypted using the public key. (page 24-25 Menezes, Oorschot, and Vanstone 1996)

   This idea is widely used within blockchain technologies. The way that the consensus network can guarantee that a transaction has been authorized by the owner of the items transacted in the transaction is based on the public key technique. This is since ownership of the items transacted over a blockchain technology network is proved by the ownership of a private key that is linked to the storage address of the items. The owner of the items can therefore authorize a transaction of the items through encrypting the transaction using the private key. Every part of the network can then easily check the authenticity of the transaction by decryption using the corresponding public key.

3. *Digital signature*: Digital signatures are used to authenticate that a message has been signed by an authorized part. The signature is done by encrypting information using a signature procedure with elements only known by the part signing the message. The signature can then be verified by a publicly known verification transformation that decrypts the encrypted message. (page 22 Menezes, Oorschot, and Vanstone 1996)

   Again this technique is used for transactions within blockchain technologies. The owner of the items transacted authenticates the transaction by signing the transaction with it's private key. The signature can then be verified using the public key known by the entire network.

4. *Time-variant parameters*: In blockchain technologies a time-variant parameter is used during the mining process called nOnce. A nOnce is a 32bit (4-byte) random number. The name nOnce refers to that the value is used no more than once for the same purpose. (page 398-

401 Menezes, Oorschot, and Vanstone 1996) For blockchain mining this means that the same value is not tried by a miner more than once for a specific block header. A incorrect number leads to a hash of the block header that does not meet the difficulty target and therefore is unnecessary to try again. As the block header includes a field for the current timestamp in form of a 32bit number that represents every second since the timestamp 1970-01-01T00:00 UTC, the block header changes every second. This gives the miners $2^{32}$ different nOnce values to try every second in order to find a correct solution.

## 2.10 Generation process

The mining algorithm is constructed to make the miners find the solution for a new block after an average determined time period. The exact time period varies between different types of cryptocurrencies. For Bitcoin this time period is set to 10 minutes. It is important to notice that this time period is an average determination, meaning that it is possible for bitcoin miners to find two blocks with only a short time span in between, while it is also possible that the time span could be extensively longer than 10 minutes.

The reward per block, is for most cryptocurrencies designed to be diminishing over time. This is the case for bitcoin. The reward for bitcoin is set to decrease by half of the previous reward after every 210000 blocks or approximately every four years. When bitcoin was first launched in January 2009, the reward was first set to be 50 bitcoins per block. In November 2012 this reward was decreased to 25 bitcoins, and in July 2016 the reward was decreased further to 12.5 bitcoins per block. This pattern will go on until approximately year 2137 when the last new bitcoin has been generated, making a total of around 21 million bitcoin, or 20.99999998 million to be exact.(Chapter 8 Antonopoulos 2014)

It is important to know that the generation process of cryptocurrencies can vary between different types of cryptocurrencies. Examples of this is that the average time per new block can vary, and the reward size per block can also vary. Even though most cryptocurrencies are designed to have a diminishing block reward over time, it is possible to design cryptocurrencies with a static, or even increasing rewards over time.(Chapter 8 Antonopoulos 2014)

After the last new bitcoins have been generated, no new bitcoins will ever be minted. After this, the only reward that miners will receive will consist of something called transaction fees. With every transaction, the sender can choose to include a fee for the miner who finds the block, hence the word transaction fee. It is not mandatory for the sender to include a fee with a transaction. However the miner who finds the solution for a block can choose not to include transactions without a fee, and will most likely prioritize transactions with fees. In order to get a transaction to be included in a block as fast as possible, it is beneficial for senders to include a transaction fee. The idea is that as the number of new minted coins per block decreases, the transaction fees will increasingly become a larger part of the total reward per block. For bitcoin, there is currently a limit for the number of transactions that can be included in a block. However this limit is artificial and will most likely change with upcoming updates to the source code of bitcoin. The idea is that if the amount of transactions per day continuously increase, the miners will still get a big enough reward from the transaction fees to still be able to profit from mining, even when no new bitcoins are generated. As the graph below shows, the number of transactions has so far continuously increased over time.(Chapter 8 Antonopoulos 2014)

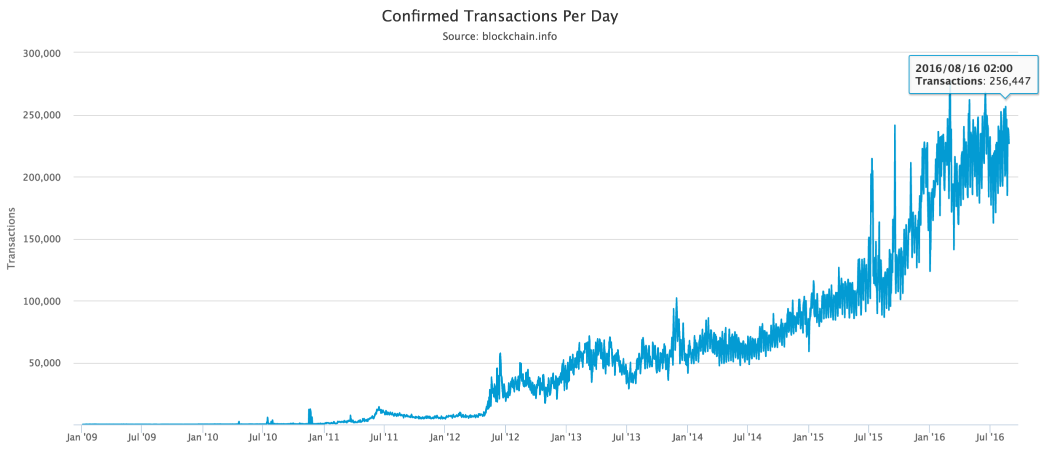


Figure 2.1: Number of bitcoin transactions over time

## 2.11 Construction of the block header

Every block has a header, that holds information regarding the block and it's construction. This header is constructed by the miner that finds the solution to the cryptographic game. The header is included in the block that the miner sends to the other miners to inform that it has found the solution to the cryptographic game. To find the solution to the cryptographic game, the miners do also need information from the previous block header. This means that a block cannot be created without its parent block, forming a chain of blocks that all are linked to each other. The header has a certain structure that is agreed upon among the miners of the network. The exact definition of the structure can vary between different types of cryptocurrencies.(Chapter 8 Antonopoulos 2014)

The bitcoin the block header has the following structure:

| Size | Field | Description |
|---|---|---|
| 4 bytes | Version | A version number to track software/protocol |
| 32 bytes | Previous Block Hash | A reference to the hash of the previous (parent) block in the chain |
| 32 bytes | Merkle Root | A hash of the root of the merkle tree of this block's transactions |
| 4 bytes | Timestamp | The approximate creation time of this block (seconds from Unix Epoch) |
| 4 bytes | Difficulty Target | The proof-of-work algorithm difficulty target for this block |
| 4 bytes | nOnce | A counter used for the proof-of-work algorithm |

Table 2.1: The bitcoin block header structure

(Chapter 8 Antonopoulos 2014)

All fields are required to be valid in order for the block to be accepted by the miners.

One of the fields holds the reference to the previous block, in the form of the hash of to the previous block. For example, the block 427139, holds the reference to block 427138, and reference it by holding its hash: 000000000000000002d18728cdf626452818bc3c621007a311d4d985c6744356 (Blockchain.info 2016a)

Another field that is of big importance is the difficulty target. Simply explained this field defines the difficulty to solve the cryptographic game. The field consists of 4 bytes, where the first byte is a hexadecimal exponent, and the other three bytes is a coefficient. In block 277316, the value for the field is 217375482757.24, which in a hexadecimal value is 329C96A385. The difficulty target is further explained in section 2.12.(Blockchain.info 2016a)

The last field holds a 4 byte nOnce. Simply explained this field is the number that the miner presents as the solution to the cryptographic game. The game is constructed in such a way that it is impossible for the miner to determine which number is correct without testing the number, but very easy to determine that a number is correct by testing it. Therefore the miners have no choice but to test different numbers randomly.

## 2.12 Mining process

The process of mining a blockchain block could simply be explained as hashing the block header continuously while changing the nOnce parameter until the resulting hash meets the difficulty target.

Bitcoin uses the SHA256 algorithm for hashing the block header. (Shirriff 2014) The type of hashing algorithm that is used varies between different cryptocurrencies. Some other examples are the CryptoNight algorithm that is used for the cryptocurrency Monero, the Scrypt algorithm that is used for Dogecoin and the Quark algorithm that is used for Quarkcoin.

A characteristic property for the different hashing algorithm is that regardless of the input length, the algorithm always produces an output of fixed length. The algorithm that is used for bitcoin SHA256, always produces as 256bit output. It is also important that the same input always produces the same result. It should also be almost impossible to find two inputs that produce the same result.

To make this more concrete, if the word Blockcain is encrypted using the

SHA256 algorithm it produces the following output:
625da44e4eaf58d61cf048d168aa6f5e492dea166d8bb54ec06c30de07db57e1 if the algorithm is implemented to return a HEX string representing the hash. As the result is completely dependent on the input, a single change to the input string, as adding an extra number in the end, completely changes the output.

To demonstrate this fact, the input string has been modified in following table, by adding an extra number to the end of the input string:

| Input | Output |
|---|---|
| Blockchain0 | 1170bfd4266d3ff0472740483fd69223e04365054e621a72ef2... |
| Blockchain1 | 0fc6e34f6899f5e2ca06688e49bb42cc104a45d5bb86c55eafe... |
| Blockchain2 | 4f07e031df167abda5623314c19c38e11358853f1c876b892a... |
| Blockchain3 | 20e39c7046b6be85eb64afacec02847fb217293cd7962b8766... |
| Blockchain4 | 2c7b2f1bcc04491890c56839d656ef80e710aeea99de670f43a... |
| Blockchain5 | 92876aecd4e88a09c69eebdc4d1c9ae2b26ea4bf5408833dc6... |
| Blockchain6 | 7de32ac537d88fb324a0d3ca4df697b0dcf8ee4be45ea5717a... |
| Blockchain7 | 442a9a086b7ffd951a03ac346e6e28729c4ab686ae38e06a14... |
| Blockchain8 | 20dd65cd4de1f6371fc6520db22ad2a85d1e8c5c12c1dadea8... |
| Blockchain9 | 10a0862fb6d82673f0256a892e8739bd146f67be28c8427b27... |
| Blockchain10 | c4cdcbe41674235a3422665ceb92146e0e8f1235e4a45c788a... |

Table 2.2: Examples of SHA256 hashing

Imagine a game where the goal is to add an extra number at the end of the input string "Blockchain" and to find an output that starts with a 0 in the beginning. As demonstrated in table above, this is the case for the input Blockchain1.

This is exactly how the mining process for bitcoin works. Instead of having the string "Blockchain" as the input string, the input is the block header as the block header data is represented as a string. And instead of adding a number to the end of the input string, the field for the nOnce repeatedly changed which completely changes the output hash. Bitcoin also uses the SHA256 algorithm, but the bitcoin mining algorithm is however implemented to encrypt the blockheader twice to produce the blockhash. (Chapter 8 Antonopoulos 2014)

As the data in the blockheader is represented as strings, the bitcoin mining algorithm is implemented to take these strings, and concatenate them before encrypting them to produce the block hash. The strings of data is concatenated in the following order:

1. Version (As hexadecimal)
2. Previous block hash
3. Merkle root
4. Timestamp (Epoch value as hexadecimal)
5. Difficulty target (As hexadecimal)
6. nOnce

The goal of the bitcoin mining process is also to find an output hash that starts with a 0. Instead of just requesting an output hash that starts with just one zero, the goal can be to find an output hash that starts with multiple zeros. How many zeros that are requested, is determined by the field for the difficulty target in the block header. (Shirriff 2014)

The bitcoin mining has over time been developed to become a little more advanced than our simple game. As the nOnce value can have around 4 billion different values, this gives the miners the ability to try around 4 billion different values per second. This is because every second, the timestamp changes which completely changes the output hash given that all the other data values for the blockheader is the same, including the nOnce value. Over time miners have however become advanced enough to try all the different nOnce values in one second. To solve this, to make sure that miners always have different values to try, the miners have been given the opportunity to modify the coinbase transaction in a block. As the merkle root has is produced based on all transactions, including the coinbase transaction, any modification to the coinbase transaction changes the hash of the merkle root. As the merkle root changes, the hash of the blockheader also changes. (Shirriff 2014)

## 2.13 Difficulty target

The function of the difficulty target is simplified to decide how many zeros the hash of a block header must start with.

The difficulty target is adjustable and varies between different blocks. The reason for this is that for bitcoin, a block should be added to the blockchain on average every 10 minutes. This is independent of how many miners at the moment are trying to find new block, and how many hashes they manage to try per second. (Chapter 8 Antonopoulos 2014)

If the difficulty target was fixed, as more miners join the mining process, an acceptable output hash would be found more and more frequently. The technology they use to compete in the mining process is also continuously developing, which would speed up the process even faster. This would quickly invalidate the idea of adding a block to the blockchain every 10 minutes.

The difficulty target is therefore adjusted frequently, to make it harder to find an accepted output hash if the number of hashes tried in the network increases. It is also modified to make it easier to find an accepted output hash if the number of hashes tried in the network decreases. This is called difficulty retargeting. For bitcoin the difficulty target is retargeted every 2016 blocks. This is done by every node in the network. How often the difficulty target is retargeted varies between different cryptocurrencies.(Chapter 8 Antonopoulos 2014) For some cryptocurrencies, like Monero, the difficulty target is retargeted every block.

For bitcoin the retargeting is done by checking how long it took for the last 2016 blocks to be added to the blockchain. This can be done by checking the timestamp field in the block header for the first of the 2016 blocks and the last, and comparing the time difference. The expected time for this is 20160 minutes. If it took less time than this, it means that the difficulty target must be increased, and vice verse if it took more time than 20160 minutes. The difficulty target is determined by the following formula:

New Difficulty = Old Difficulty * (Actual Time of Last 2016 Blocks / 20160 minutes) (Chapter 8 Antonopoulos 2014)

If a new block is proposed by a miner after the retargeting that has not adjusted the difficulty target, it will not be accepted by the other miners and will therefore not be added to the blockchain.

## 2.14 Mining distribution

Historically, in the early years of bitcoin mining, bitcoin miners used the CPU of their computers to participate in the mining process. However as more and more miners started participating, competition rose. This made miners use better and better hardware in order to have an edge over their competitors, hence making the difficulty target rise. This is since the better hardware a miner has, the more nOnce per second can be tried. More tried nOnces per second makes the chance to find a correct solution higher and therefore increases the overall reward for a specific miner over time. (Chapter 2 Antonopoulos 2014)

The raising difficulty target, made miners use more and more specialized mining hardware. Soon miners realised that they could try more nOnce values per second (also called hashes per second or hash/s) using the GPU of computers with high end graphic cards, such as gaming computers. (Chapter 2 Antonopoulos 2014)

In 2013 a new type of bitcoin mining hardware emerged. The new hardware had the bitcoin mining algorithm (SHA256) printed directly in the hardware. This specialized hardware is called ASIC (for: application-specific integrated circuits). ASIC:s proved to be even more effective than GPU mining. Most GPU units produce less than 1 GH/s (one million hashes per second) (Wiki 2016a), while many ASIC units produce more than 1000 GH/s while consuming less electricity. (Wiki 2016b)

At the time of writing, a miner needs to use ASIC:s in order to profitably mine bitcoin. This is since the value of the reward a miner can produce with even the most effective graphic cards, doesn't exceed the cost of the electricity it consumes. (Antonopoulos 2014)

For many other cryptocurrenices than bitcoin, this is however not necessarily the case. As ASIC:s are highly specialized a specific ASIC unit can only be used to hash inputs using the specific algorithm it has printed into the hardware. This means that another cryptocurrency that doesn't use the same hashing algorithm as bitcoin cannot be mined using a specialized bitcoin ASIC unit.

For many types of cryptocurrencies, there are therefore no developed ASIC units to mine them. This makes these type of coins much more efficient to mine with a desktop computer. There are also cryptocurrencies that use algorithms that are specialized to be effective for CPU mining.

## 2.15 Mining pools

Mining cryptocurrency is a highly competitive industry. At the time of writing (august 2016) the hashrate of the entire bitcoin network is around 1,6 billion GH/s. (Blockchain.info 2016b) A miner can buy an ASIC unit for example producing 4730 GH/s for around 1000 $ dollars (august 2016). (Bitmain 2016) If a miner buys this specific ASIC unit, that miner can produce a hash rate equal to around 0,000394% of the entire bitcoin network. That means that while mining, the miner would find the correct solution to approximately every 253700 block on average, which is equal to around 4,83 years. As the unit also consumes 1293 watt, the miner also have to take the power consumption over the time period into calculation.

This equation is however based on the assumption that the hashrate of the entire bitcoin network stays the same during this entire time period. This will most likely not be the case, as historically the hashrate of the bitcoin network has been increasing rapidly as time progress (Blockchain.info 2016b). As the hashrate of the entire network increases, the difficulty target is also recalculated which makes it harder to find the correct solution with the current hardware the miner is using.

This makes mining for a single miner with a hashrate equal to a very small percentage of the bitcoin network very luck based. A miner has the same probability to find the correct solution for the next block independently if that miner has found the correct solution for the previous 3 blocks or haven't found the correct solution for 4 years. This means that over the time span of 4,83 years, the miner could be extremely lucky and find 10 correct solutions, or end up not finding the correct solution for a single block over the entire time period.

At the seldom occasion that the miner finds the correct solution, it is highly rewarded. The current reward is 12.5 bitcoins valued at approximately 7200 $ dollars with the current bitcoin price (August, 2016), plus any miner fees included in the transactions of the block.

For most miners it is preferable to split this reward to small payments over time than to receive it all in one huge payment. This makes mining rewards more reliable and easier to calculate. For this reason mining pools have been invented.

Mining pools basically makes it possible for multiple miners to collaborate to try to find the solution for a block, and to split any reward received between the miners. If any miner of a mining pool finds the correct solution to a block, how much of the reward the different miners receive is based on how much contribution they have made to find the block. The idea is to fairly distribute the reward between miners that mines with different hardware, as some miners produces much more hashes per seconds than others. The mining pool usually also take a small percentage of the block reward for the coordination of the miners. Mining pools also make sure to split the work into different solutions that they want the connected miners to try, to make sure that multiple miners connected to the pool are not trying the same solutions, and do the same work more than once. Currently the vast majority of all the hashing power of the bitcoin network goes through mining pools (Blockchain.info 2016c).

To calculate how much contribution a miner has made, the difficulty target to get a share of the reward from a pool is much less restrictive than the actual block header difficulty target, often more than 1000 times easier. This means that more nOnce values will be correct to get a share of the block reward. When a miner submits a solution that meets the difficulty target set by the pool, this is called that the miner submits a share. (Chapter 8 Antonopoulos 2014)

Occasionally, a miner will provide solution that meets both the difficulty target set by the pool, and the actual difficulty target of the blockchain. When such an event happens, the pool publishes the block, and distributes the reward from that block to the different miners based on how many shares they have contributed with. The mining pool usually pays the miners the reward they have received, as soon as it has exceeded a given amount. (Chapter 8 Antonopoulos 2014)

Technically, a miner connects to a mining pool with a specialized mining software that communicates through a given protocol. The mining pool provides all connected miners solutions to try. If a miner finds a solution that meets the requirements set by the pool difficulty target, the miner submits it to the pool. The pool notes that the miner has submitted a share, to calculate how much reward the miner will get once a block is found by the pool. The mining pool is a full node that is connected to the rest of the network. This makes all miners restart the mining process again. And the process to try to find the solution to the next block in the blockchain is started.

If a miner outside of the pool finds the solution to a block and broadcasts it before the pool does, the pool informs all the miners connected to it to stop mining the previous block. The pool will instead provide the miners with a new block that the miners will try to find a solution for.

Mining pools communicate with the miners through a specialized protocol for cryptocurrency mining, called Stratum.

## 2.16 Stratum

In the past, the communication between mining pools and miners was done through the Hypertext Transfer Protocol (HTTP). However as HTTP was designed for website browsing where the client asks for specific content, using the HTTP for mining was not optimal. Using HTTP for pool mining created three main problems:

As the amount of hashes a single miner could produce increased rapidly, the network load on the mining pools also steadily increased. One request from the miner to get the solutions that the pool wanted the miner to try (also called one getwork job) could last for one 4.2 GH/s mining rig for around one minute, or until a new block was found by the rest of the network. As many miners have hardware that generates much more GH/s than that, a pool received tremendous amount of HTTP getwork requests. A pool often has several thousand connected miners, which puts much stress on the pool. (SlushPool 2012)

Another problem was that the miners needed to have regular contact with the pool in order to make sure that the solutions they were trying were for valid blocks. This is to avoid committing invalid shares, hence getting them rejected. Having a regular contact with the pool did however lead to much network load. To solve this the Long polling pattern was used. The Long polling pattern is a known solution for managing real time updates for regular web technologies. As the situation is different for cryptocurrency mining, it is however not optimal. (SlushPool 2012)

After a long polling broadcast, the clients need to reconnect to the server. When many miners did this during the same time period it could be hard to distinguish actual miners trying to reconnect from DDoS attacks. (SlushPool 2012)

The situation when miners communicate with pools is different compared to normal HTTP client web browsing. The situation is turned around, and instead the server already knows large parts of the content they need to provide to the miner. (SlushPool 2012)

To solve the given situation with its problems, the Slushpool (a bitcoin mining pool) came up with a solution using a new protocol called Stratum. This protocol has now been implemented for a majority of the cryptocurrency mining pools (even though it may have some adjustments for different cryptocurrencies). (SlushPool 2012)

The Stratum protocol doesn't use HTTP for the communication. Instead the Stratum protocol uses a plain TCP socket connection to transfer JSON-RPC messages in the payload containing the information between the miners and the pool. (wiki 2016)

This approach brings some benefits. One benefit is that as the messages are JSON-RPC messages, they are in a human readable form. This simplifies debuggning, as the programmer can read the communication between the client and the pool.(SlushPool 2012)

The JSON messages are transferred through a TCP socket, and typically fit in one TCP segment. This means that the overhead is not as high as a HTTP message, even though the overhead possibly could be reduced even further through the use of other protocols. Using JSON objects the code also becomes easy to extend without invalidating backwards compatibility. JSON, or Javascript objects are widely supported on all platforms.

The biggest benefit however is that the server can drive the load themselves. There is no need for alternative solutions to work around the problems with HTTP. This means that the Long polling solution is not needed as the server can transfer messages to the miners at any time. This also reduces the problem with distinguishing miners trying to connect to the server, from

DDoS attacks. (SlushPool 2012)

Another big benefit is that the new protocol can increase the amount of possible solutions the miners can try when mining a block. This is especially important for bitcoin as the hashing power of the network has increased rapidly. This has increased the difficulty target to a point where the old solution just doesn't provide the miners with enough solutions to try for a block. (SlushPool 2012)

The reason that Stratum provides the miners with an increased amount of solutions to try, is that the Stratum protocol allows the miners to produce unique block heads locally. Before Stratum the miners where only allowed to iterate over the time and nOnce field of a block header. With Stratum the miner can also edit the coinbase transaction field by adding the nOnce in it. The coinbase transaction holds the address to where the blockreward will be sent. This coinbase transaction field can however be modified without breaking anything. The coinbase transaction was before set by the mining pool (server-side), but with the Stratum the miners are free to modify this (client-side). When modifying the coinbase transaction, the merkelroot for the block changes. This means that when hashing the block header, network load on the server is reduced as miners can try more solutions without communicating with the server. (SlushPool 2012)

Stratum transfers only merkle branch hashes to the miners, instead of a complete dump of server's memory pool as the old method did. Stratum also scales much better to the increasing amount of transactions in the bitcoin network. This is since the miners themselves don't have to deal with all the transactions, but leaves that job to the pool. (SlushPool 2012)

Lastly, Stratum also made the process of checking if the shares from the miners are correct much less demanding for the pool.

*Other alternatives*: Instead of using JSON, there are other protocols that possibly could have been used. However these alternatives includes a few disadvantages which has further affected the distribution of the Stratum protocol among the pools within the different cryptocurrencies.

Some of the possible alternatives are:

1. *Custom text protocol*: Custom text protocols comes with the benefit of being human readable and therefore easy to debug, just like JSON.

However as the text protocol is custom, this also means that all the predefined features in JSON, such as request, responses and serialization has to be defined, which would seem unnecessary as JSON already has these features defined. (SlushPool 2012)

2. *Custom binary protocol*: The most beneficial reason for using a custom binary protocol would be that the network load would be lower than using JSON, as there is some overhead for sending JSON-RPC messages. However as binary data is not human readable, debugging the communication is harder than JSON objects. Developing serializers and deserializers for binary data is also challenging. As JSON already has a predefined serializer it can be argued that this is an unnecessarily tricky approach. (SlushPool 2012)
3. *Protocol buffers*: Protocol buffers is an technique developed by Google. The technique could possibly work just as well as JSON. However Google has only protocol compilers for C++, Java and Python. This is not that beneficial as cryptocurrencies software, such as miners, often are implemented in other languages, such as C for performance reasons. (Google 2016a) There are however third party implementations for various other languages, such as C, C#, Haskell, Perl and Ruby.(Google 2016b)

## 2.17 Stratum methods

The communication between the client and the pool is sent through JSON messages with a specific structure. An example of such a message that is used for bitcoin mining pools is:
"id": 1, "method": "mining.subscribe", "params": []\n (wiki 2016)

In this case this is a message sent from the client to the pool in order to subscribe to the pool. The id field of the JSON message holds the id of the miner that is connecting. The method field holds the information of the action the miner wants to do, in this case it is to subscribe to the pool. The params field holds extra information as parameters to be sent into the method. In this case the field is empty. The \n after the JSON message is mandatory in order to signal the end of the JSON-RPC message.

An example response that the pool can send to the previous request is:
{"id": 1, "result": [ [ ["mining.set_difficulty", "b4b6693b72a50c7116db18d6497cac52"], ["mining.notify", "ae6812eb4cd7735a302a8a9dd95cf71f"] ], "08000002", 4], "error": null}\n

The id field of this messages corresponds to the request id field. The result field holds the response information corresponding to the request information. The last two fields in the result contains the extranOnce1 the miner will use (in this case "08000002"), and the number of bytes the miner will use for the extranOnce2 counter (in this case 4). Lastly, the error field holds possible errors that the request has lead to.

There are several different methods that the client can send to the server, and the exact implementation varies between different cryptocurrencies. For bitcoin mining the methods from the client to the pool is:

1. **mining.subscribe("user agent/version", "extranOnce1")**
   The mining subscribe method is used by the client to connect to the pool, in order to initialize the mining process. The method has two arguments. The first argument "user agent/version" is set to the clients miner version. The second parameter "extranOnce1" is optional. It allows the miner, if supported by the pool, to send an extranOnce to use as the starting point. This may be used if the miner for example has lost connection to the pool when mining using a specific nOnce. If so, the miner can resume its mining process with that nOnce. (wiki 2016)

2. **mining.extranOnce.subscribe()**
   Indicates to the server that the miner supports that the server sets a extranOnce starting point. (wiki 2016)

3. **mining.authorize("username", "password")**
   The authorize method is used to authorize the miners pool account. This means that if the pool is implemented to only allow miners that have created accounts on the pool, the username field must correspond to that account. Some pools may however be implemented to create a new account with the username if such an account does not exist. The pool often sets the password field as optional. If not, it must correspond to the created account. Worth noting is that there can often be several usernames connected to one account. An example of this

is that if the username for an account is "Satoshi", two different usernames called "Satoshi.miner1" and "Satoshi.miner2" can exist. This allows two different miners to be connected to the same account, to allow any rewards from shares that these miners submits to be sent to one account.

The response from the pool to an authorize call is either true or false, corresponding to if the username and password (if not optional) is accepted by the pool or not. (wiki 2016)

4. **mining.get_transactions("job id")**
   Gets the associated transactions that should be included in a block for a specific job id. The response will include the transactions in hex format. (wiki 2016)

5. **mining.submit("username", "job id", "ExtranOnce2", "nTime", "nOnce")**
   This is the method the miner will use when submitting a share for a specific job. The miner will call this method when it has found a hash that meets the difficulty target set by the pool. When the miner has submitted the share, the pool will respond with true if the share is accepted, and false if it is not. The username field should be set to the miner worker name. The job id field should correspond to the id of the current job. The ExtranOnce2 field should have the size set by the pool through the response to the miners mining.subscribe call. The nTime field should be the current timeStamp. The nOnce field should correspond to a nOnce the miner has found that leads to a hash that meets the pool difficulty target. (wiki 2016)

6. **mining.suggest_difficulty(preferred share difficulty Number)**
   This is a method that can be used by the miner to suggest the difficulty target that the pool will use as the boundary that hash shares must meet. The pool is not required to take this suggestion into consideration, even if the pool supports this feature. (wiki 2016)

7. **mining.suggest_target("full hex share target")**
   The miner can suggest a share target of the pool by using this method. This is often done before the miner calls the mining.subscribe method. The pool is not required to take this suggestion into consideration, even if the pool supports this feature. (wiki 2016)

There are also several different methods the pool can use to send information from the pool to the client. The exact implementation varies between different cryptocurrencies, but the stratum protocol for bitcoin supports the following methods:

1. **mining.set_difficulty(difficulty)**
   Used by the pool to set the difficulty target that the hashes provided by the miners must meet in order to get an accepted share. The miners should implement this new target when they receive their next job. (wiki 2016)

2. **mining.notify("Job ID", Hash of previous block, Generation transaction (part 1), Generation transaction (part 2), List of merkle branches, Bitcoin block version, nBits, nTime, Clean Jobs)**
   This method is used by the pool to assign the miner to a job. The miner will then try different solutions in order to try to find a solution that leads to a hash that meets the difficulty target set by the pool. The information for the different parameters should be:
   *The "Job ID"* field should be set to a unique id that the miners will use when sending back information such as a share to match the information to a specific job.
   *The Hash of previous block* field is necessary for the miner in order to build the block header.
   *The Generation transaction (part 1)* field is used by the miner by inserting the ExtranOnce1 and ExtranOnce2 after the information provided by this field, in order to build the block header.
   *The Generation transaction (part 2)* field holds the information the miner will insert after the part 1 field and the inserted ExtranOnce1 and ExtranOnce2 in order to build the block header. This completes the full generation transaction that the miner provides.
   *The List of merkle branches* field is used to build the merkle root, by hashing the full generation transaction with the merkle branches.
   *The Bitcoin block version* field holds the block version, which is needed to build the block header.
   *The nBits* field holds the encoded information of the difficulty target of the entire network (not the pool difficulty target), this is used when building the block header.
   *The nTime* field holds the current time stamp.

*The Clean Jobs* field is used to inform the miners if they should move on to the next job or not. An example of when this could be appropriate would be when the bitcoin network has already found the solution to the block the miner has a job assigned to. This would make any new share to that block useless. If true, the miner is informed to move on to the next job immediately. If false, the miner will move on once the miner has tried every possible nOnce solution for the current job it is assigned to. (wiki 2016)

3. **mining.set_extranOnce("extranOnce1", extranOnce2_size)**
   This method will be called by the pool in order to replace the values sent to the miner in the initial response to the miner.subscribe call from the miner. This will happen as a response if the miner has called the mining.extraonce.subscribe method.(wiki 2016)

4. **client.get_version()**
   Is called in order to ask the miner to send its miner name and version.(wiki 2016)

5. **client.reconnect("hostname", port, waittime)**
   This method is called in order to ask a client to reconnect to the pool. The hostname field is the url the miner should reconnect to. The port field holds the port the miner should use when reconnecting. The wait time field contains the number of seconds the miner should wait before reconnecting after it has disconnected. The miner may ignore this request if the hostname and port isn't the same as it is currently using. (wiki 2016)

6. **client.show_message("human-readable message")**
   This method is called in order to send a message to the miner that it should present to the user. The message should be inserted in the "human-readable message" field. (wiki 2016)

# Chapter 3

# Computational micropayments

## 3.1 Mining reward as micropayments

Many different types of cryptocurrencies have a value. While it has been widely discussed whether cryptocurrencies has any intrinsic value or not as they are not backed up by any state, it is hard to argue against the fact that they are valued at the price that people are willing to pay.

Currently the value of many of the most popular and used cryptocurrencies are most often measured in fiat currencies. At the time of writing, bitcoin the most popular cryptocurrency is valued around 575 $ dollar (August 4th 2016). The value of many other less used cryptocurrencies is instead often measured by how much bitcoin one unit of that specific type of cryptocurrency can be traded for. One example is the cryptocurrency Monero. At the time of writing (August 4th 2016), one unit of Monero can be traded to around 0.0032 bitcoin, on a specific exchange platform that allows users to trade different cryptocurrencies for other types of cryptocurrencies. One example of such an exchange platform is Poloniex (https://poloniex.com/).

The reward for finding a solution of a block for a cryptocurrency consists of a given amount of that type of cryptocurrency. That means the reward will have a value given that the specific cryptocurrency has a value. Mining pools allow the miners that have submitted shares to the mining pool to split the reward of a block found by the pool. This provides a situation where a share to a mining pool has a value. Most mining pools set the difficulty target to submit a share to much lower than the actual difficulty target of the block. This leads to that the value of one submit is only a fractional part of the full block reward value.

In this paper, we will provide a design theory that concludes how these fractional parts of the full reward can be used to create micropayments through the computational power of a user's computer.

## 3.2 Structure of solution

In order to build a system that allows fractional parts of the mining reward to serve as micropayments, we suggest some crucial design decisions that allow such an approach.

First of all the miners must connect to a mining pool, in order to split the full reward into fractional parts that will form the micropayments. The system must therefore be structured to allow users to make micropayments through mining clients that connects to a mining pool. As the fractional reward that the user receives is the actual micropayment, it is important that the reward is not transferred to the user. Instead it should be transferred to the part that acts as the receiver of the micropayment.

The design of the system that we propose also allows multiple users to combine their fractional mining reward, hence micropayment, to the same receiver. The reason for this is that one micropayment of for example 0,1 USD cent has a too insignificant value for a receiver. However if 1 million users combine their individual micropayment of 0,1 USD cent, the combined value reaches 1000 USD dollars.

Mining of cryptocurrency is not naturally designed for this goal. First of all the mining softwares are not designed to make the user make a limited amount of submits to reach a specific value. Instead the mining softwares are designed to let the clients mine continuously and make continuous submits until the user itself decides to turn the mining software of. The mining software is also designed to make the user manually setting for the miner, such as the pool username that the reward will be transferred to.

*Architecture of theoretical framework*: We suggest a design that instead of just the individual mining clients and the mining pool also includes a server that acts a proxy between the clients and the mining pool to solve the issues described above.

The idea with the server is to provide the mining pool with the information of where the mining reward should be sent. This is to make sure that all reward of the miners that are making micropayments to the same receiver will be transferred to the same output address. The server will also provide the miners with the information of how many submits the miners have to contribute with to meet the value of the micropayment. When the client has generated enough value, it will terminate the miner process. The server will also keep a record of how many submits a miner has made, in order to know how much value a specific client has generated.

Steps of theoretical proposal:

1. *Miner connects to proxy server*: The client connects to the proxy server through a TCP-socket connection.

2. *Proxy server sends mining information to miner*: The proxy server then sends the necessary information to the client. This contains information for the clients miner, such as which algorithm to use and how many submits the client needs to provide.

3. *Miner sends Stratum JSONs*: The miner will then start sending JSON objects following the Stratum protocol to the proxy server. The proxy server will then check the JSON objects and replace necessary information such as output address of the mining reward. After this the proxy will transmit the possibly modified JSON objects to a mining pool.

4. *Mining pool sends back a Stratum response*: The pool will then send back a JSON object following the Stratum protocol as a response. The proxy server will check this response and take necessary actions. The actions includes keeping track of the amount of accepted submits the miner has made if the pool responds that a submit the miner has made has been accepted. After this the proxy server will transmit the JSON object to the miner.

5. *Step 3 and 4 repeated*: The steps 3 and 4 are repeated until the miner has made enough submits to have generated enough value for the mi-

cropayment. When this occurs, the mining process of the client is terminated.

To make a concrete example, imagine that a popular blogger posts information regarding technical updates in the cellphone industry. To finance the time spent writing the blog posts, the blogger has given readers the possibility to donate 1 cent worth of their computational power to the blogger. The reader Bob who wants to donate 1 cent, runs the client program which receives information of what algorithm to use from the proxy server. The proxy server also sends how many shares Bob's miner should submit. The proxy server decides that the algorithm to run is the CryptoNight algorithm to mine the cryptocurrency Monero as it is the most profitable cryptocurrency to mine at the time the reader wants to donate money. The proxy server connects the miner with MoneroPool, a Monero mining pool and calculates how many submits Bob's mining program needs to commit. The reward for finding the solution to one block is set to around 12 Monero which has a total value of around 96 dollars at the time of writing (28th, August 2016), with a block difficulty target set to around 2,500,000,000. The difficulty target for an accepted share at the given pool is however only set to 25000, making it 100,000 times easier to find an acceptable hash for the pool then the actual block. This makes a share to the pool valued on average 100,000 times less than an acceptable solution for the entire block. The value is therefore 96 USD dollar / 100,000 which is equal to around 0,1 USD cent per share. This means that in order for Bob to make a micropayment of 1 cent he must make 10 submits. The miner then keep trying to find acceptable shares by running the hashing algorithm on the jobs provided by the pool through the proxy server, until 10 acceptable shares have been made. When this happens, the miner will terminate and Bob will have made his full donation of 1 USD cent.

In the following sections of this chapter, the different architectural parts of the system will be explained more in detail, and their responsibilities.

## 3.3 Client

The main responsibility of the client-side of the solution is to run a miner for cryptocurrency.

We suggest to make the client-side system to either be a standalone program that the user can preferably install on its computer or a system that runs in the user's web browser. A web browser system is suggested to be an add-on that the user can install to their web browser. Both of these approaches implies advantages unique to the chosen approach.

The main advantage of choosing to make the client-side system as a standalone program is that it is very easy to create a system that runs in the background. This can be done without substantially effecting the performance of the user's hardware when the user is doing other tasks on the computer simultaneously. The advantage to have the program run in the background is that the user specifically must ask to terminate the specific standalone program in order to make it stop mining, if the user wants to terminate it before having made enough submits. This is not the case for the web browser approach as the mining process is terminated as soon as the web browser is closed by the user. It is suggested that the miner of the standalone program only uses the current free resources of the hardware when it is running in the background. It should scale down as soon as the user starts other processes that requires resources of the hardware.

Choosing the approach to run the system in a web browser mainly gives the advantage of creating a smooth user experience when the user is going to start the mining process. Lets say a vendor places a donate button on their web page to let users donate 1 cent worth of their computational power. When the user presses this button, the system can start directly without requiring the user to interact with another program outside the browser. This requires that the web browser mining add-on is set up.

Another important aspect is if the mining process should use the CPU or the GPU. There are different hashing algorithms, and also different cryptocurrencies, for both of alternatives. It is also possible to have two separate miners running at the same time with different hashing algorithms. If so its possible to target both the CPU and GPU at the same time to maximize the value generated per minute. It is important to take into consideration that an average desktop computer can compete better with other miners if it is using the CPU. This is because the difference in performance between an average desktop computer's CPU and an optimized computer is less significant than than the difference of the GPUs. It is also important that maximizing the performance of the GPU will limit the user experience more when the user will try to use the hardware for other tasks simultaneously, compared to the CPU. An example is that the GPU generates a lot more heat when being pushed to its limit, compared to the CPU.

## 3.4 Proxy Server

The proxy server of the theoretical framework acts like a middle man between the client system and the mining pool by receiving all communication from the miner (in the form of Stratum JSON messages) and then transmitting them to a mining pool. When the pool answers, the JSON response is transmitted back to the miner.

The basic layout for this is simple. As the the communication between the miner and the pool is based on the Stratum protocol, it is humanly readable and can therefore be easily interfered and handle accordingly by the proxy server.

How different messages should be handled depends on the specific message and type of algorithm being run by the miner. There are some general messages that are suggested to be handled independent of this:

1. *Login request*: The first communication that will be sent from the miner is a login request. We suggest that the parameters for the login name and possibly password (if required) is changed in this message to correspond to the receivers information of the micropayment. By adapting this approach, the receivers' information is never really disclosed to the client. In some cases this can be beneficial for the receiver as they might not want to make this information public. This also makes it possible to create a system where the login name of the miner is unique for every client before the name is changed, in order to uniquely identify the different miners.

2. *Job*: The pool will then answer by replaying with a job that the miner should try to find the solution for. None of this information has to be manipulated in any way, and can be transmited back to the miner directly.

3. *Miner sends submit*: If the miner finds a solution to a job, it will send

the solution back to the pool as a share. This message doesn't really have to be handled in any other way then being transmitted to the pool, unless the proxy server should check if the solution is valid or not.

4. *Pool sends submit status*: The pool will then answer if the share was accepted as a valid solution or not. If accepted the proxy server should store the information that the miner has made a correct share in some way, and then transmit the message to the miner. It is important to only store the submit as a valid share when the pool responds that it was accepted, and not immediately when the miner sends the submit. This is since the client-side of the miner can be manipulated to send incorrect solutions often in order to try to reach the amount of requested submits faster.

5. *Miner resends login request*: If the miner doesn't receive a new job from the pool after a given amount of time, the miners are also programmed to resend a log in message. It is important that the proxy server is programmed to be able to handle this correctly, in order to not handle the request as a request from a new miner. If this happens, the proxy server should not set the submits made by the miner when sending the log in message again to 0.

The proxy server also has other responsibilities than the handling the stratum messages. Other important responsibilities includes:

1. *Provide the client with an algorithm to run*: It is suggested that the proxy server is programmed to be able to find out which cryptocurrency is the most profitable to mine in terms of value per minute for the hardware of an average desktop computer. This limits the time the client has to spend running the mining process. This probably maximizes the amount of clients that will submit all required shares. When a cryptocurrency has been chosen, the server should send information to the client of which hashing algorithm it should run in order to mine that specific coin.

2. *Provide the client with amount of submits to reach*: The pool should also provide the client with information on how many accepted shares it should submit before the micropayment is complete. This is to make sure that the client can turn off the mining process once the target has

been hit.

3. *Keep track of accepted submits*: The pool should also keep track of how many accepted submits the miner has made. This is since the miner can reconnect several times and shouldn't have to start over with the submit count every time. As the client-side also can be manipulated by the end user, it is not viable to store this information on the client-side. This makes it possible to implement functionality for a user to use several different computers, but to still submit shares for the same micropayment.

## 3.5 Mining Pool

When designing the system there is a key decision that has to be made regarding the choice of mining pool. It is important to decide whether a designated mining pool should be set up specifically for the system or if a third party mining pool should be used. Both of these alternatives have benefits and disadvantages.

*Designated mining pool*: The benefit of using a designated mining pool or pools is mainly to have the control of the pool. This removes the risk of having to trust a third party, which is quite important since the cryptocurrency that will be mined has a value. Also the need to pay a fee to the third party for using the pool is removed. However managing a pool is not simple, and involves much complexity which is an obvious disadvantage. Another disadvantage is that the pool requires many connected miners at the same time, in other words users using their computational power for micropayments. Statistically the overall reward per hash should be the same regardless of how many miners a pool has. As fewer miners means that a correct solution will be found more seldom, this also means that receivers possibly have to wait an extended amount of time before they can withdraw any value from the pool. It is therefore highly recommended that pools with a significant amount of the total hashing power of the network for a specific cryptocurrency are used, as this will lead to continuous payments often.

*Third party mining pool*: Key benefits of using third party mining pool or pools include the fact that it makes it easy to build a system that allows clients to mine the most profitable cryptocurrency. It is easy to find functioning mining pools for pretty much every major cryptocurrency. There are also mining pools that can be used to mine several different cryptocurrencies in the same pool. It is also positive that the complexity of managing a mining pool is removed. However the most important disadvantages of this approach is that the mining pools will then charge a small fee for every share, hence taking a small part of every micropayment. Also there is complexity involved in using third party mining pools while enabling withdrawals of the receivers of the micropayments cryptocurrency. This is because withdrawals work differently in different mining pools. On top of this, using third party mining pools adds risk as they have to be trusted to hold the receivers cryptocurrency.

Our recommendation based on this is to use third party pool or pools. The exception is if the amount of users using the system is high enough to host pools that produce hashing power that is a significant part of the network of the cryptocurrency being mined. If so a dedicated pool or pools for the system is recommended.

# Chapter 4

# Functioning prototype

In this chapter, the prototype that has been created will be explained in detail. The previous chapter explained the theoretical approach to a solution for using computational power as micropayments. This chapter will instead explain a developed functioning prototype that proves that the theoretical approach works.

Worth to take into consideration is that the developed prototype is only a proof of concept, and not a optimized solution. This means that there are several parts of the prototype that need to be further developed in order to make the prototype function in a full scale business system.

The chapter is divided into several different sections, each explaining the implementation of the different parts of the prototype. Lastly the chapter includes a section that describes further developments of the prototype that would be needed in order to make it function in a business perspective.

## 4.1 Client

In our prototype the client is a standalone Windows Form C# program. The idea of the program is to be a minimalistic program that runs in the background without effecting the performance for the other tasks that the user runs simultaneously.

The user interface has been made as simple as possible, and its only function is to display the progress of the micropayment. This has been done by only including the most necessary information to the user. Because of this the user has not been given any ability to interact with the user interface other then terminating the program.

The reason for making a standalone program was to prove that the client program can run in the background while the user is using the computer for other purposes. The program will function without effecting the performance for the other tasks that the user runs simultaneously.

The miner that the client use has been limited to only use the CPU and not the GPU. Part of the reason for this solution is that the using the CPU is less noticeable for the end user than using the GPU. Another reason for this limitation is that the developed client system is made to only function as a prototype and not as an optimal solution. Using the GPU adds much complexity when programming the system. This is because the settings for the miner system must be based on the type of graphic card that the desktop computer uses. To develop a client system that handles this correctly without requiring manual set up of the settings is complex. This was the main reasons for limiting the prototype to the CPU, as the prototype was made to show a proof of concept while keeping the developing process as simple as possible.

For simplicity reasons the client miner has not been developed specifically for this project, instead a third party open source miner has been used. Because of this the part of the client system developed in the prototype is the GUI that interacts with the miner program without the need for the user to manually edit anything. The third party miner that is used is called "Minerd". The prototype client system is limited to only mining one type of cryptocurrency, Monero.

As the GUI has been developed as a C# Windows Forms application, it is currently run as an .exe file which limits the systems usage to the Windows operation system. This limitation is not optimal for every type of user. This limitation was made to remove the complexity of developing a cross platform system.

## 4.2 Proxy server

The prototype proxy server has been developed as a C# console application. The way to connect to the proxy server is through TCP-socket connections from the client application. The proxy server will then set up a TCP-Socket connection to a mining pool specifically for that client and transmit necessary JSON messages between the miner and the pool.

The pool allows asynchronous TCP-connections, which is highly preferable as this means that a client won't block a thread to the proxy server when waiting for messages from the pool.

Ideally the proxy server would be set up on a virtual machine using the Microsoft Azure environment, as this is a robust hosting alternative that scales based on the traffic the proxy server is currently handling. However since the system is only meant to function as a proof of concept prototype, it is currently only hosted locally to prove its functionality.

The pool is connected through a TCP-Socket connection to the DNS that the pool is hosted on. As Stratum is based on JSON messages sent over TCP, the proxy server does not allow HTTP connections, and is therefore not a public HTTP-API.

A very important aspect is that the prototype proxy server is able to handle several connected miners at the same time without high latency. It is also able to track individual information of the separate miners, such as number of accepted shares.

One aspect of the prototype proxy server that is not ideal is that the proxy server cannot decide the most profitable cryptocurrency to mine for the time being. It is set up to only let the connected clients mine the cryptocurrency Monero. To minimize the time spent for the clients to generate enough value, this is solution not optimal.

The decision not to include this feature is because it is straightforward how to develop this feature, but requires more development time than available for the prototype. What is required is that the proxy server would need to connect to API:s of block explorers for the different cryptocurrencies it should evaluate. This is to acquire information regarding the amount of hashing power for the networks of the different cryptocurrencies, as well as the different difficulty targets. It is also required to connect to an API that can provide information of the price of one unit of the different cryptocurrencies. With this information it is possible to calculate how much the hashing power of an average desktop computer is worth per minute, for the different cryptocurrencies. With this information it is easy to calculate the currently most profitable cryptocurrency to mine for the users.

## 4.3 Mining pool

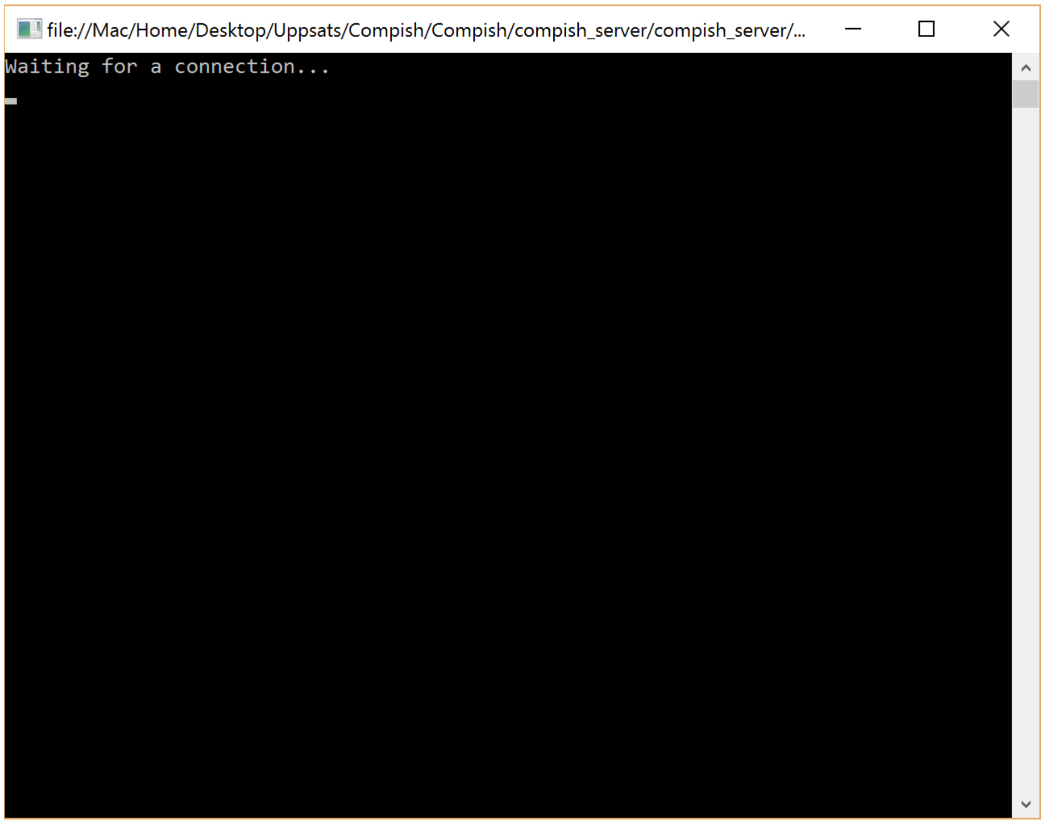
As the developed system is a prototype it will not have several thousand connected users. For this reason, it is not preferable to use a specified mining pool for the system. This leads to the decision that third party mining pools is the most viable solution to use.

As the prototype is only a proof of concept, only one type of cryptocurrency will be mined by the client miners. For this reason only one mining pool has been used. This mining pool is a Monero mining pool with the descriptive name MoneroPool.com.

## 4.4 Demonstration of prototype

In this section we will demonstrate the developed prototype by demonstrating and explaining images of the prototype in action.

*Proxy server is started*:
The first action is the initialization of the proxy server, which is a console application. In a real functioning business system the proxy server would be hosted on a virtual machine using Microsoft Azure. Using Azure, the console application would run continuously, and would only be started once.


file://Mac/Home/Desktop/Uppsats/Compish/Compish/compish_server/compish_server/...

```
Waiting for a connection...
```

Figure 4.1: The proxy server is started

Important to notice is that the proxy server implementation has for the sake of this demonstration been modified to print events and communication that goes through it. In a real functioning business system, this would not be necessary.

*Client is started*:
A client is started and is connected to the server. The client then sends a login request following the Stratum protocol to the proxy server, which will be sent to a mining pool. Note that the login name the miner sends to the proxy server is not the login name that will be sent to the mining pool. The proxy server will modify the login name. The login name should correspond to the micropayment receiver's information.

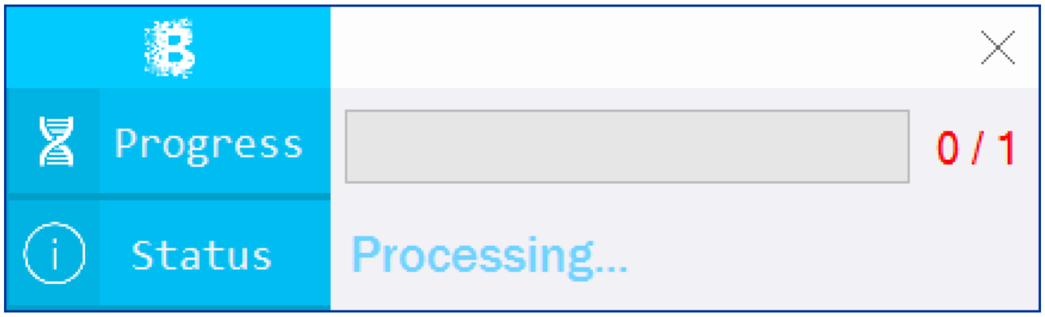
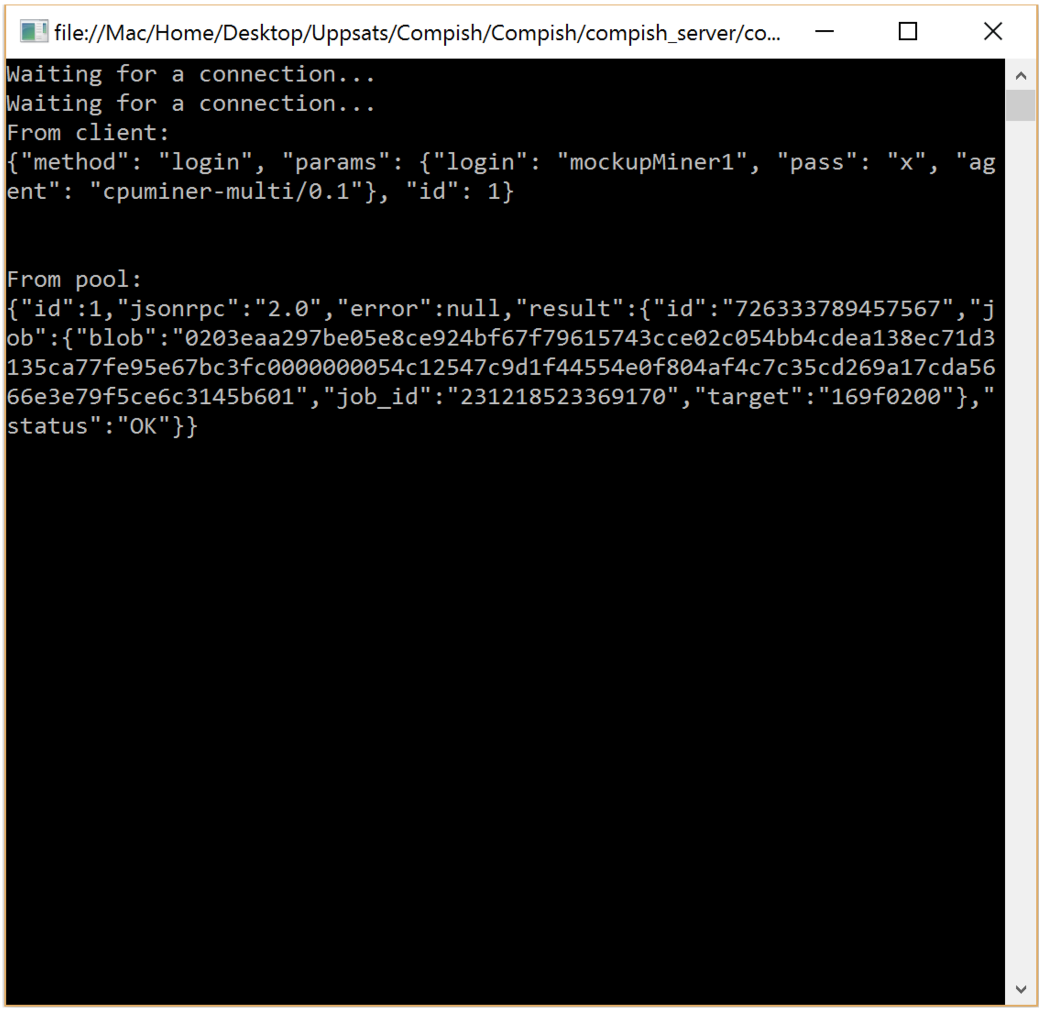


Figure 4.2: The client program is started

*Job is received from pool*:
The mining pool responds by sending back a job to the proxy server, that the client's miner should start finding a solution for. The job is sent through a Stratum JSON message, that is transmitted back to the client through the proxy server.


file://Mac/Home/Desktop/Uppsats/Compish/Compish/compish_server/co...

```
Waiting for a connection...
Waiting for a connection...
From client:
{"method": "login", "params": {"login": "mockupMiner1", "pass": "x", "ag
ent": "cpuminer-multi/0.1"}, "id": 1}


From pool:
{"id":1,"jsonrpc":"2.0","error":null,"result":{"id":"726333789457567","j
ob":{"blob":"0203eaa297be05e8ce924bf67f79615743cce02c054bb4cdea138ec71d3
135ca77fe95e67bc3fc0000000054c12547c9d1f44554e0f804af4c7c35cd269a17cda56
66e3e79f5ce6c3145b601","job_id":"231218523369170","target":"169f0200"},"
status":"OK"}}
```

Figure 4.3: The job is recived and sent back to the client

*Client receives job*:
When the client receives the job, the miner will start trying to find the correct solution by hashing the block header using the blob sent by the pool. If the hash meets the difficulty target set by the target field in the job JSON received from the pool, the miner will submit back the solution found. In the prototype, the client will signal that it is working to find a solution by displaying a small blinking loading bar in the GUI. Another more appropriate way to display this could of course be implemented.

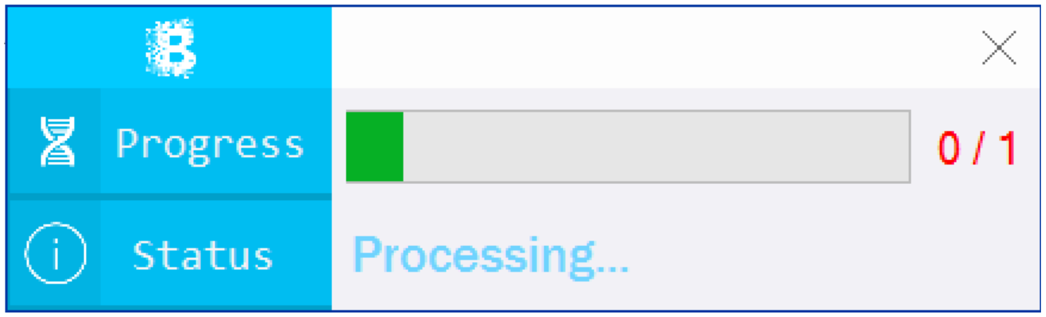
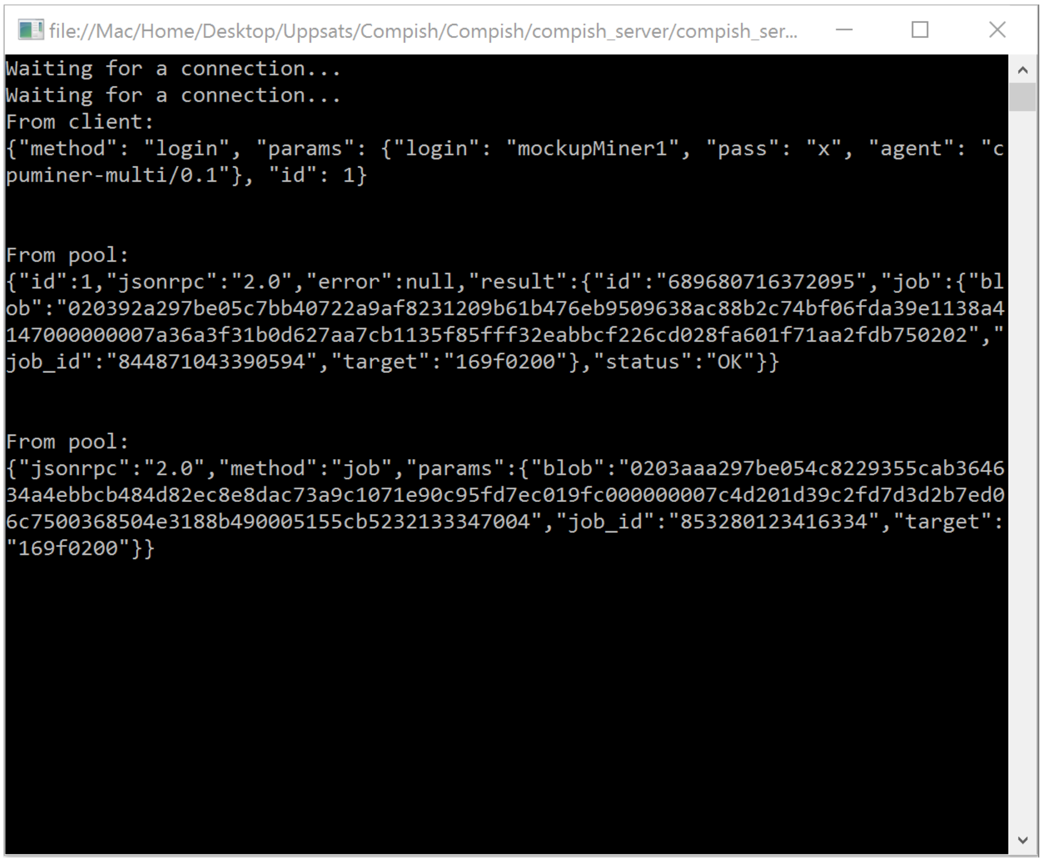


Figure 4.4: The job is received and sent back to the client

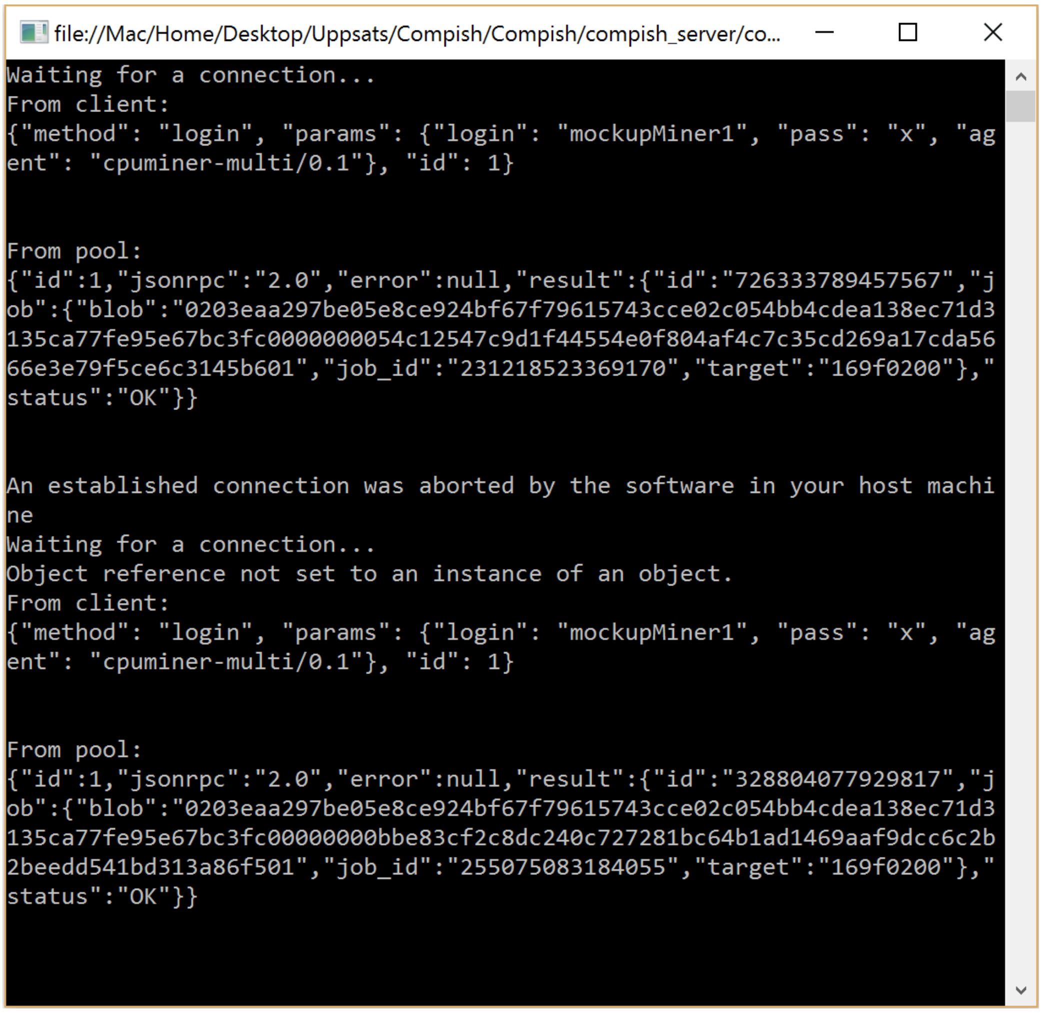
*New block found*:
If a solution for the current block is found by the Monero network, all miners should abandon their current jobs and start trying to find a solution for the next block. When this happens, the pools sends a new job to the proxy server that is transmitted back to the client which will start working with the new job.


file://Mac/Home/Desktop/Uppsats/Compish/Compish/compish_server/compish_ser...

```
Waiting for a connection...
Waiting for a connection...
From client:
{"method": "login", "params": {"login": "mockupMiner1", "pass": "x", "agent": "c
puminer-multi/0.1"}, "id": 1}


From pool:
{"id":1,"jsonrpc":"2.0","error":null,"result":{"id":"689680716372095","job":{"bl
ob":"020392a297be05c7bb40722a9af8231209b61b476eb9509638ac88b2c74bf06fda39e1138a4
147000000007a36a3f31b0d627aa7cb1135f85fff32eabbcf226cd028fa601f71aa2fdb750202","
job_id":"844871043390594","target":"169f0200"},"status":"OK"}}


From pool:
{"jsonrpc":"2.0","method":"job","params":{"blob":"0203aaa297be054c8229355cab3646
34a4ebbcb484d82ec8e8dac73a9c1071e90c95fd7ec019fc000000007c4d201d39c2fd7d3d2b7ed0
6c7500368504e3188b490005155cb5232133347004","job_id":"853280123416334","target":
"169f0200"}}
```

Figure 4.5: A new job is sent to the miner

*Client reconnects*:
The miner will sometimes send a new login request to the pool, and will then receive a new job. If this happens it is important that the submit count is not reset either on the client or the proxy server.

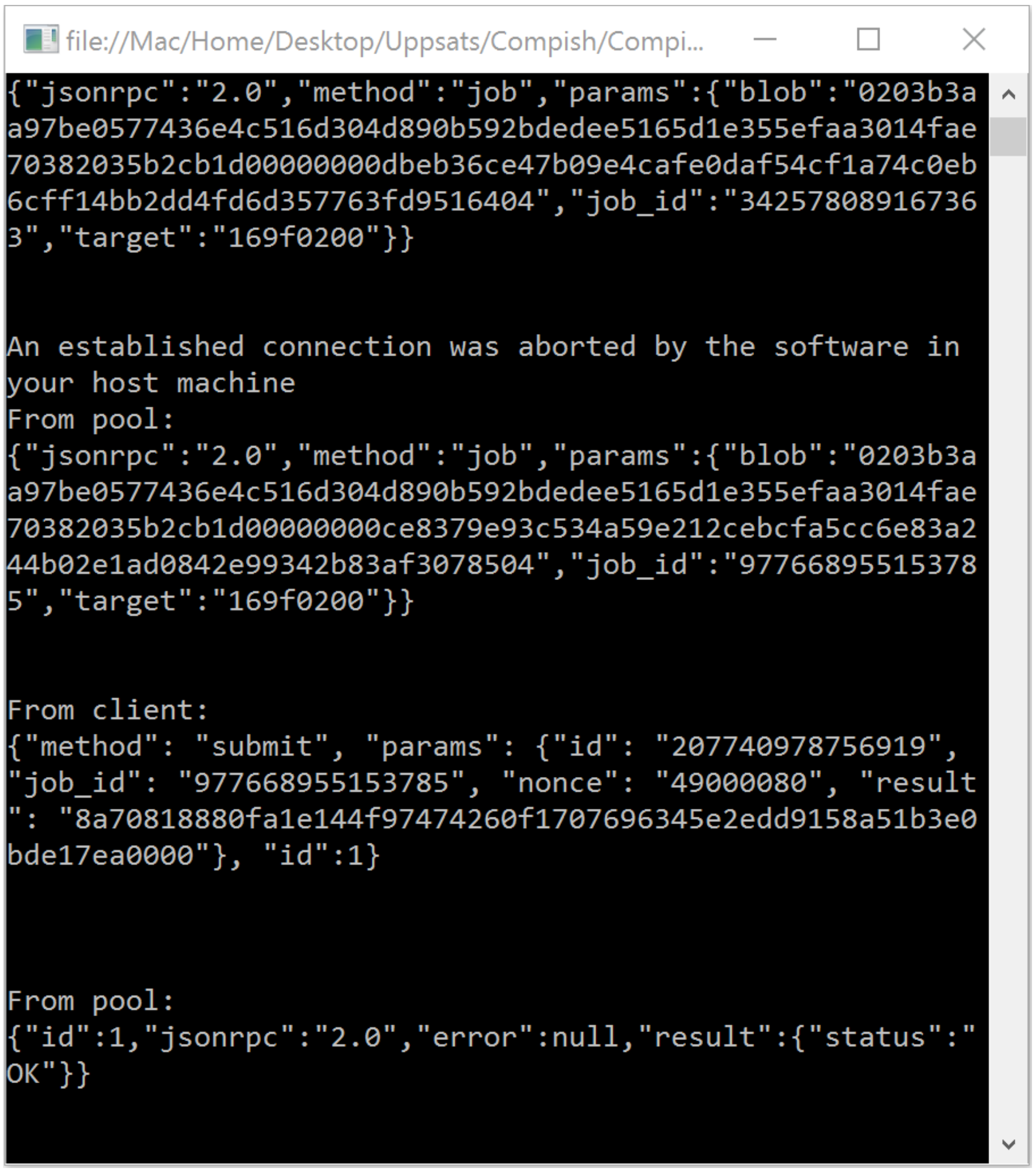

file://Mac/Home/Desktop/Uppsats/Compish/Compish/compish_server/co...

```
Waiting for a connection...
From client:
{"method": "login", "params": {"login": "mockupMiner1", "pass": "x", "ag
ent": "cpuminer-multi/0.1"}, "id": 1}


From pool:
{"id":1,"jsonrpc":"2.0","error":null,"result":{"id":"726333789457567","j
ob":{"blob":"0203eaa297be05e8ce924bf67f79615743cce02c054bb4cdea138ec71d3
135ca77fe95e67bc3fc0000000054c12547c9d1f44554e0f804af4c7c35cd269a17cda56
66e3e79f5ce6c3145b601","job_id":"231218523369170","target":"169f0200"},"
status":"OK"}}


An established connection was aborted by the software in your host machi
ne
Waiting for a connection...
Object reference not set to an instance of an object.
From client:
{"method": "login", "params": {"login": "mockupMiner1", "pass": "x", "ag
ent": "cpuminer-multi/0.1"}, "id": 1}


From pool:
{"id":1,"jsonrpc":"2.0","error":null,"result":{"id":"328804077929817","j
ob":{"blob":"0203eaa297be05e8ce924bf67f79615743cce02c054bb4cdea138ec71d3
135ca77fe95e67bc3fc00000000bbe83cf2c8dc240c727281bc64b1ad1469aaf9dcc6c2b
2beedd541bd313a86f501","job_id":"255075083184055","target":"169f0200"},"
status":"OK"}}
```

Figure 4.6: Login request resent by miner

*Solution found*:
When the miner finds a correct solution that meets the difficulty target of the job, it will send in the submit to the proxy server. The submit is then

transferred further to the mining pool. If the mining pool responds with an "OK" status, the submit was a valid share.

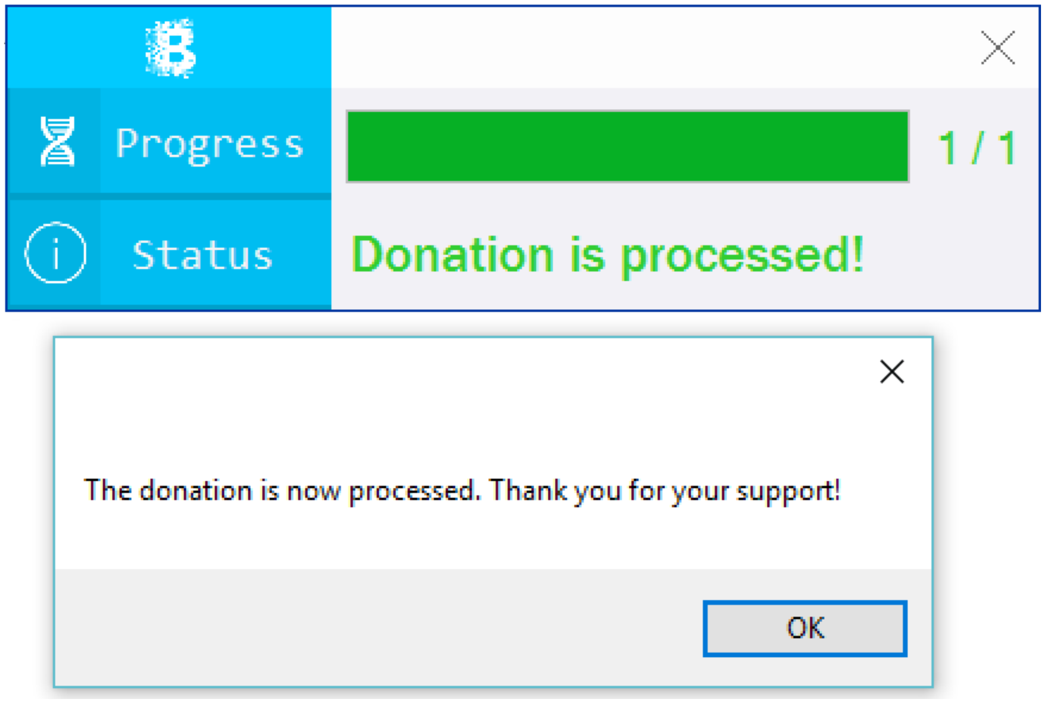


file://Mac/Home/Desktop/Uppsats/Compish/Compi...

```
{"jsonrpc":"2.0","method":"job","params":{"blob":"0203b3a
a97be0577436e4c516d304d890b592bdedee5165d1e355efaa3014fae
70382035b2cb1d00000000dbeb36ce47b09e4cafe0daf54cf1a74c0eb
6cff14bb2dd4fd6d357763fd9516404","job_id":"34257808916736
3","target":"169f0200"}}


An established connection was aborted by the software in
your host machine
From pool:
{"jsonrpc":"2.0","method":"job","params":{"blob":"0203b3a
a97be0577436e4c516d304d890b592bdedee5165d1e355efaa3014fae
70382035b2cb1d00000000ce8379e93c534a59e212cebcfa5cc6e83a2
44b02e1ad0842e99342b83af3078504","job_id":"97766895515378
5","target":"169f0200"}}


From client:
{"method": "submit", "params": {"id": "207740978756919",
"job_id": "977668955153785", "nonce": "49000080", "result
": "8a70818880fa1e144f97474260f1707696345e2edd9158a51b3e0
bde17ea0000"}, "id":1}



From pool:
{"id":1,"jsonrpc":"2.0","error":null,"result":{"status":"
OK"}}
```

Figure 4.7: Miner submitted valid share

*Client receives information of valid share*:
When the message is sent to the miner that it has submited a valid share, the submit count is increased. When the submit count reaches the required amount to have generated the value for the micropayment, the mining process is aborted. When this happens, the client is alerted that the micropayment is complete. The client program is closed when the user closes the alert message. Note that for the sake of this demonstration the required submit count has been set to 1. However for a larger micropayment, this count would be set to a higher number.



Donation is processed!

The donation is now processed. Thank you for your support!

Figure 4.8: Micropayment is complete

## 4.5 Future developments of prototype

The developed prototype functions as a proof of concept. There are however some extra functionality that would be preferable to add in order for the prototype to be viable in a business perspective. These functionalities would be preferable for a business that wants to add the possibility to receive micropayments through computational power.

Our idea of how the prototype could be viable in a business perspective requires the development of two extra functionalities:

1. *Payment option on website*: In order for a business to add the possibility to receive payments though computational power, they must be able to add some control that lets the user download the client system and start the mining process. The control should also determine the value of the micropayment that the users will generate. We have two suggestions for such controls. For businesses that have full access to edit the HTML content on their website, we suggest that the business should be able to add a button that displays a simple message such as "donate 1 cent of your computational power". Once a user presses the button, the client system will be downloaded along with the correct settings. The settings should be corresponding to the business that will receive the value. If the user that clicks the button already has the client system downloaded, only the settings for the receiver will be downloaded.

   For businesses that don't have full access to edit the HTML content on their website (such as business operating on social medias, Youtube and similar platforms), another solution is required. Our solution is that the business should be able to generate a hyperlink that leads those clicking on the link to download the client system. The system will then install with the correct settings for the specific receiver of the micropayment.

2. *Registration for receivers*: If a business wants to receive micropayments in cryptocurrency the business needs to somehow set up an account at a mining pool as well as a cryptocurrency wallet address to link their account to. However this process may be far to complex for most businesses, especially if they lack knowledge in how cryptocurrency func-

tions. For many businesses receiving the payment in cryptocurrency is also not optimal as all their utilities, staffing salaries and etc. needs to be payed in a fiat currency.

For these reasons our solution is to develop a website that lets businesses register with all the necessary information. Our website system will then handle all connections with mining pools so that the business does not need to set up their own wallets. When the business later wants to withdraw the value from the micropayments, they do not need to operate through mining pools. The suggestion is instead to let the business operate through our website. The businesses should then have the opportunity to choose if the value should be withdrawn as cryptocurrency or as a fiat currency. If the business choose to withdraw the value as a fiat currency, all the cryptocurrency that users have generated through micropayments will be exchanged to fiat currency on an exchange platform.

## 4.6 Previous implementations

When we started writing this thesis and developing the prototype, there was to our knowledge no released implementation that used the computational power of desktop computers to generate micropayments.

The closest implementation to our knowledge was the 21 bitcoin computer. The 21 bitcoin computer can be connected to a desktop computer through a USB-port. The computer can allow micropayments through mining, if used to develop such a service. The 21 bitcoin computer is however fundamentally different from our solution. The 21 bitcoin computer is a designated specialized hardware for bitcoin, and can be used to mine bitcoin effectively. It can however not be used to mine other cryptocurrencies. The 21 computer has been developed to allow developers to develop applications for bitcoin, and is not mainly developed to allow micropayments through mining. (21.co 2016)

During the development of the prototype and the writing of the thesis, the user "ctorres" posted a developed system that is quite similar to our solution on the Bitcointalk forum. The post was made the 14th July 2016. The system is called "TicketMiner" and allows users to run a web browser add on to generate value. There is one fundamental difference to our proposal. The proxy server is meant to be hosted by the receiver of the micropayment. This requires that the receiver hosts a node.js server. From a business perspective we argue that this in most cases is not viable, as it requires knowledge of the receiver to host a server, and requires knowledge in node.js for maintenance. This also means that it is not usable if the receiver does not have full access to edit the HTML content. It eliminates the easy set up and interaction that our prototype can allow if it is further developed, but could potentially allow micropayments through cryptocurrency mining if set up correctly. (ctorres 2016)

# Chapter 5

# Concluding remarks

## 5.1 Conclusions

In this thesis we have developed a design theory to show how cryptocurrency mining can be used to generate value to make micropayments.

The relevant background knowledge for our design theory has been described in chapter 2. In this chapter we in detail explained the underlying technology behind cryptocurrency, as well as briefly explaining the previous research and concepts that cryptocurrency and blockchain technology are based on.

Further we have proposed a theoretical framework that applications to realize micropayments through cryptocurrency can be based upon in chapter 3. We have concluded that in order to do so the system needs to include three essential parts. A client system, a proxy server and a mining pool or pools. We have described the essential parts of these systems. We have also described which parts that need to be developed and which parts that can use third party solutions.

Chapter 4 describes a prototype that is based on this theoretical framework proposed in chapter 3. This prototype was developed to show that a real world application based on the theoretical framework is functional. By developing a functional prototype, we have concluded that it is indeed possible to generate value for a micropayment through the use of cryptocurrency mining. In accordance with Gregor and Jones a developed design theory must consist of 8 core components. We do here conclude that our developed design theory does so:

1. *Purpose and scope*: The purpose and scope is described in chapter 1 of this thesis. The chapter explains that the purpose of this thesis is to develop an artifact to make it possible to use cryptocurrency mining as a way to generate value for micropayments. The chapter also explains that the scope is to develop a theoretical framework that describes how this can be done, as well as a prototype built upon this theoretical framework.

2. *Principle of form and function and Constructs*: In chapter 3 we describe a theoretical framework of how cryptocurrency mining can be used for micropayments. In this chapter we clearly define the architecture of the framework, as we define three main architectural components, the client side, the proxy server and the mining pool. We also define clear constructs of the different components, such that the client side should include a miner that can run cryptocurrency mining algorithms.

3. *Artifact mutability*: The theoretical framework described in chapter 3 allows artefactual mutability as it is a generalizable framework and does not define an exact solution other than its three core components. The actual implementation of the framework can vary which is also discussed. A mentioned example is that the client system could either be a standalone system, or a browser add on. It is also mentioned that third party solutions can be used for parts of the system, as well as the client miner can either be using the GPU, CPU or both.

4. *Testable propositions*: Chapter 3 also defines several testable propositions of the design theory such as: Can the proxy server transmit the JSON messages between the pool and the miner? Can the mining pool provide the miners with work? And maybe most importantly, can the system generate value for micropayments?

5. *Justificatory knowledge*: We define relevant background knowledge in chapter 2. This chapter describes the past research that cryptocurrency is based on, such that the cryptography and the byzantine generals problem.

6. *Principles of implementation*: The theoretical framework described in

chapter 3 clearly defines guidelines, that can be followed to develop a system that allows micropayments through cryptocurrency mining.

7. *Expository instantiation*: Chapter 4 describes a developed prototype that is based on the theoretical framework described in chapter 3. The prototype is a proof of concept to prove that applications can be built based on the theoretical framework. This verifies that cryptocurrency mining can be used to generate value for micropayments.

## 5.2 Discussion

We claim that the theoretical framework that we have developed in this thesis is valid and functional to base and develop a full scale system upon. Further we claim that since the theoretical framework is valid and functional, it proves that the value generated when mining cryptocurrency can be used as micropayments.

The most valid proof we have for this claim is the fact that we have developed a functioning prototype, and we claim that this prototype is a valid proof of concept. The reason for this is that the prototype (even though it is not yet developed into a full scale system) clearly can allow multiple clients to generate value. This value can be transferred to a single wallet address that is decided by the proxy server.

This claim does however raise a few concerns that must be addressed in order to be valid.

1. *Insufficient number of connected clients tested*: One concern that is important to discuss is the fact that the prototype has not been tested with an extensive amount of users, hence connected miners that generate a value. Since one of the core ideas of the system from a business perspective is that an extensive amount of connected clients together will generate a value that is significant to the receiver, one can claim that the prototype is not a proof of concept until this has been done. This is not the case with the prototype, as it has not been tested other than locally. As it has been tested locally, it has not been possible to let extensive amount of clients on separate hardware connect to the proxy server. Instead only multiple miner clients started on the same

hardware have been able to connect to the proxy server. This means that all clients are sharing the same hardware. This in turn means that all clients are only generating as much total hashing power as if there was only one single client running on the hardware.

We do however claim that this concern is not an issue as all procedures used when the system is running for two connected clients are exactly the same as if there are one million connected clients for the same receiver. The only difference is the amount of resources the system requires, which increases as the amount of connected clients increases. As the system is also based on asynchronous programming, users will not block threads while waiting for responses from the proxy server. The specific implementation of the prototype also allows it to be hosted using Microsoft Azure, which scales the resources depending on the need.

Something important to notice is that despite the amount of connected clients, value is always generated when using the prototype. This is since the prototype uses a third party pool which has many more connected miners that are not connected to the prototype. This means that as long as a client is able to make a valid share to the pool, it will generate value. The only thing that is different with fewer connected clients to the prototype is that the total value for the receiver will be lower as fewer clients will make micropayments. This may lead to that the total value generated will not be enough if the receiver wants to withdraw the value as a fiat currency as the transactions fees of fiat currency might be too high. This also applies to some types of cryptocurrencies if the total value generated for the receiver is very low. This does not invalidate the fact that value is generated to a given receiver, hence creating a micropayment using cryptocurrency mining. It is just too low to be of any valid use for the receiver.

It is also important to notice that all technical implications that comes with a higher connected client amount are implementation concerns that can be solved programatically and with extensive testing. Such concerns is out of the scope of this thesis. The thesis focuses on providing a design theory with a theoretical framework that describes how micropayments based on cryptocurrency mining can be achieved, not the unique way of implementing the theoretical framework.

2. *One type of cryptocurrency mined*: A concern where the developed prototype partly contradicts with the proposed theoretical framework is that the framework suggests that the system should be able to identify the most profitable cryptocurrency to mine. The system should then provide the clients with this information, and let the clients mine that cryptocurrency to optimize the amount of time the client needs to spend during the mining process. However the prototype has been limited to only handle one type of cryptocurrency. As the prototype only needs to show the proof of concept, we argue this is an acceptable limitation. The value needed for the micropayment will take longer to generate compared to mining the currently most profitable cryptocurrency, but it will still be generated after a given amount of time. We claim that this is enough as a proof of concept as the aim of the thesis is to prove that cryptocurrency mining can be used as micropayments, not to find the most optimal way to do so. Also the theoretical framework only suggests this as an optimal solution, not as a requirement to build a functioning system.

3. *More clients leads to rising difficulty*: If more users start using the system, this would lead a increased total hashing power for the network of the cryptocurrency they are mining. This leads to an increased difficulty and making it more difficult to find a correct solution for a block for that type of cryptocurrency.

   One could argue that if everything else stays the same, the hashing power per user will then become less and less valuable as more hashing power is competing for the same amount of reward. The system would then become counterproductive as it would take longer and longer time for a specific user to generate enough value for their micropayment as the system gets more users.

   However this argument has been proven to be wrong time after time for several different cryptocurrencies. History has shown that as mining becomes less profitable, either because of dropping reward value or increased competition, many miners simply tend to stop mining and therefore lowering the total hashing power of the network. The reason for this is simple, most miners run the mining process to make a profit. If they are not profitable anymore, which could be because the electricity costs exceed to reward they receive, they have no reason to keep mining.

We therefore claim that this would be the most likely scenario if this became an issue for an implementation of the system. This guarantees that the system will still be viable, even if the system became used by a majority of the population.

Important is also that its possible to implement a system that makes different users of the system mine different cryptocurrencies, hence spreading the hashing power among the networks of different types of cryptocurrencies.

4. *Fewer cryptocurrencies can be mined in the future*: An argument that would invalidate the use of the design theory is that the amount of cryptocurrencies with a actual value may decrease in the future. This is because there could instead be only a few dominating cryptocurrencies that holds all the value. This would most likely lead the development of specified hardware to mine all of valuable cryptocurrencies. As we have already stated, when there is specified hardware to mine using a specific algorithm, a desktop computer generates value at a much slower rate as the hashing power it can produce is very low compared to the specified hardware. This makes mining on desktop computers very unprofitable, and would make a system built based on the design theory next to unusable.

   However history seems to prove that this is an incorrect assumption. This is because the amount of different types of cryptocurrencies with a actual value and the types of hashing algorithms they are using are increasing, not decreasing.

   We also believe that this development can be explained since the development of cryptocurrencies is the first time where different currencies can have unique properties that gives them an advantage in a niched area. For example, one cryptocurrency can be niched to be the most anonymous cryptocurrency, while another cryptocurrency can be niched to be the most transparent cryptocurrency. This is different from the fiat currencies, where the actual properties of the currencies do not vary, other than the state behind them and their monetary policies.

   For this reason we speculate that in the future, the amount of different cryptocurrencies with a value will continue to raise.This would sustain the validity of the design theory.

It is also much easier to publish a new cryptocurrency using a new hashing algorithm, than to build specified hardware that can only be used to mine using a specific mining algorithm. This is since it's expensive to develop and produce specified hardware. A producer must be able to produce and sell or use the specified hardware in order to generate a higher value than the development and production cost.

## 5.3 Future research

The design theory that has been developed in this thesis can potentially be developed further in many ways and lead to much further research. This section lists two areas that we believe would be very interesting to focus future research within.

*Possibility for very small value payments*: The entire idea of the thesis is to create a new way to do micropayments. However the word micropayment is quite vague and subjective as to how small the value of a payment has to be to be called a micropayment. Some people can subjectively think that a payment of 1 $ dollar is a micropayment, where others would think that it needs to be under 1 $ cent. In this thesis we define micropayments as a transaction valued at 1 $ cent or less. However the solution proposed in this thesis allows for much smaller micropayments than this. It would for example allow a micropayment valued at 0.00001 $ cent. This requires that the system uses a mining pool which has set the difficulty at a low enough level to let one share be valued at this value.

A large scale system to allow such small value transfers that is widely used does not currently exist. The transaction costs for fiat currencies exceed this small value drastically to allow such small payments. Even for bitcoin, the most popular cryptocurrency, the transaction fee for a transaction currently averages around a value of 3 cents. This makes transactions of a lower value unprofitable.

A system built based on our theoretical framework could potentially fill this gap. This could possibly open up for payments for content that before has been impossible as they have had a too low value to monetize. It's hard too imagine what such content would be as it would be a completely new market area. However two potential examples could be wallpaper images on a website that before distributed them for free, or a website that runs an algorithm on your profile picture for social media to automatically set the lightning settings of the image to preferable levels.

It would be very interesting to do further research on this subject. The research could focus on what this new payment channel could lead to, and the potential new market areas it could open up.

*The system can be generalized to compute other algorithms*: The design theory in this thesis is focused on providing a solution that describes how to allow micropayments through cryptocurrency mining, as this is the scope of the thesis. However the solution could be even more generalized. If the theoretical framework is more generalized there are possible solutions that would not require cryptocurrency mining to generate a value.

If for example the mining pool was instead replaced with an institution that required much computational power to solve an issue and they were willing to pay for it. If so, the problems that the clients would run an algorithm to try to solve could be changed to the institution's problems. An easy example to think of would be if a university was interested of running an algorithm to find new prime numbers. As long as the proxy server could communicate with the university and it could scale up different solutions to try among the miners this would be a very viable system to do so. By this modification, the system has then allowed micropayments through computational power without using cryptocurrency. This requires that the university would be willing to pay a fee to the micropayment receiver depending on how much computational power the client contributes with. Note that the problem to solve could of course be changed to anything else that would require much computational power.

This more generalized solution as well as different problems that it could be used to solve, would be interesting to research further.

# Bibliography

Leslie Lamport, Robert Shostak and Marshall Pease (1982). “The Byzantine Generals Problem”. In: *ACM Transactions on Programming Languages and Systems* 4.3, pp. 382–401.

Menezes, Alfred J., Paul C. van Oorschot, and Scott A. Vanstone (1996). *Handbook of applied cryptography.*

Shirley Gregor, David Jones (2007). “The Anatomy of a Design Theory”. In: *Journal of the Association for Information Systems* 8.5, pp. 312–335.

Nakamoto, Satoshi (2008). “The bitcoin white paper”. In:

SlushPool (2012). *Startum Mining Protocol.* URL: `https://slushpool.com/help/#!/manual/stratum-protocol`.

Antonopoulos, Andreas M. (2014). *Mastering Bitcoin: Unlocking Digital Cryptocurrencies.*

Nicolas T. Courtois, Marek Grajek and Rahul Naik (2014). *The Unreasonable Fundamental Incertitudes Behind Bitcoin Mining.*

Shirriff, Ken (2014). *Bitcoin mining the hard way: the algorithms, protocols, and bytes.* URL: `http://www.righto.com/2014/02/bitcoin-mining-hard-way-algorithms.html`.

21.co (2016). *21 bitcoin computer.* URL: `https://21.co/buy/`.

Bitmain (2016). *Bitmain.* URL: `https://enshop.bitmain.com/productDetail.htm?pid=000201511170341298180m44675v0613`.

Blockchain.info (2016a). *Bitcoin blockchain block 427139.* URL: `https://blockchain.info/sv/block/000000000000000001ad749dc81bcb3cf4d63a5cb38e8154abd648b`

– (2016b). *Bitcoin network hashrate.* URL: `https://blockchain.info/charts/hash-rate`.

– (2016c). *Bitcoin network pools.* URL: `https://blockchain.info/sv/pools`.

ctorres (2016). *TicketMiner*. URL: `https://bitcointalk.org/index.php?topic=1550230.0`.

Google (2016a). *Protocol buffers*. URL: `https://developers.google.com/protocol-buffers/`.

– (2016b). *Protocol buffers other languages*. URL: `https://developers.google.com/protocol-buffers/`.

Wiki, Bitcoin (2016a). *Hardware comparison of non specialized hardware*. URL: `https://en.bitcoin.it/wiki/Non-specialized_hardware_comparison`.

– (2016b). *Hardware comparison of specialized hardware*. URL: `https://en.bitcoin.it/wiki/Mining_hardware_comparison`.

wiki, Bitcoin (2016). *Stratum mining protocol*. URL: `https://en.bitcoin.it/wiki/Stratum_mining_protocol`.