

Difficulty Adjustment Algorithms for Preventing Proof-of-Work Mining Attacks

by

Hamid Azimy

Master of Computer Science, University of Tehran, 2015

**A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF**

Doctor of Philosophy

In the Graduate Academic Unit of Computer Science

Supervisor(s): Ali A. Ghorbani, Ph.D., Faculty of Computer Science
Ebrahim Bagheri, Ph.D., Electrical & Computer Engineering
Department, Ryerson University

Examining Board: Rongxing Lu, Ph.D., Faculty of Computer Science
Scott Buffett, Ph.D., National Research Council and Faculty
of Computer Science
Donglei Du, Ph.D., Faculty of Management

External Examiner: Xiaodong Lin, Ph.D., School of Computer Science,
University of Guelph, Canada

This dissertation is accepted by the
Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

August, 2022

© Hamid Azimy, 2022

Abstract

Bitcoin mining is the process of generating new blocks in the blockchain. This process is vulnerable to different types of attacks. One of the most famous attacks in this category is *selfish mining*, introduced by Eyal and Sirer [21] in 2014. Selfish mining is a very well-known attack and many studies tried to analyze, mitigate, or extend it. This attack is essentially a strategy that a sufficiently powerful miner can follow to obtain more revenue than its fair share. To put it simply, it works by slowing down the network and wasting the hash power of both attackers and honest miners but wasting honest miners' hash power more. This attack is not exclusive to Bitcoin and can be performed on many proof-of-work blockchains and cryptocurrencies (e.g. Ethereum) and it is observed in a few cases on other altcoins (Monacoin).

The reason that selfish mining is effective in Bitcoin is the difficulty adjustment algorithm in Bitcoin. Because after the difficulty adjustment, the selfish miner will benefit from higher relative revenue. This is the point that is not well-studied in the literature and we try to address it in this thesis. However, the difficulty adjustment algorithm is an essential part of the Bitcoin protocol and cannot be removed. In this thesis, we analyze the profitability of selfish mining concerning time and also the presence of other selfish miners. We also propose a family of alternative difficulty adjustment algorithms including Zeno's DAA, Zeno's Max DAA, and Zeno's Parametric DAA, that discourage selfish mining, while allowing the Bitcoin network to remain scalable (by adjusting the difficulty of the network).

We analyze our proposed solutions, using two methods: mathematical analysis and simulation analysis. Then, we present the results and discuss the effectiveness of our proposed solutions. Based on our analysis, our proposed algorithms effectively increase the profitability waiting time for the attackers to almost double its original value. For example, for a miner with 40% of the network's hash power, it extends the waiting time from four weeks to more than eleven weeks. This will discourage attackers from performing their malicious activities. We also show that our proposed algorithm allows the network to scale while it increases the waiting time.

Dedication

I wish to dedicate my thesis to my beloved wife, Nastaran.

This wouldn't have been possible without her endless support and encouragement.

I will be forever grateful to have her by my side during this journey.

Acknowledgements

I would like to express my gratitude to my great supervisors, Dr. Ali Ghorbani and Dr. Ebrahim Bagheri. Working with them and learning from them was a special opportunity for me. I truly appreciate their help and guidance throughout my years of study. Also, I am so thankful to be a part of Canadian Institute for Cybersecurity (CIC) family and get to know many amazing people.

Table of Contents

Abstract	ii
Dedication	iv
Acknowledgments	v
Table of Contents	vi
Table of Contents	vi
1 Introduction	1
1.1 Problem Background	1
1.1.1 Overview of Bitcoin	1
1.1.2 Difficulty Adjustment	3
1.1.3 Selfish Mining	5
1.1.4 Importance of Selfish Mining	6
1.1.5 Blockchain Trilemma	7
1.2 Motivations and Contributions	8
1.3 Thesis Organization	10
2 Background and Literature Review	12
2.1 Bitcoin Preliminaries	12
2.1.1 Identities	12
2.1.2 Transactions	13

2.1.3	Bitcoin Mining	16
2.1.4	Pooled Mining	17
2.2	Blockchain and Bitcoin Threats	18
2.2.1	Privacy and Anonymity Issues in Bitcoin	18
2.2.2	Security of Bitcoin	21
2.3	Selfish Mining	23
2.3.1	Detection and Mitigation	28
2.4	Summary	30
3	Methodology and Proposed Solutions	32
3.1	Profitability of Selfish Mining	32
3.1.1	Empirical Evidence	33
3.1.2	Profitability Metrics	34
3.1.3	Findings	36
3.1.4	Profitability of Selfish Mining in Time	39
3.2	Desirable Properties of a DAA	43
3.3	Alternative Difficulty Adjustment Algorithms	46
3.3.1	Zeno’s DAA	48
3.3.2	Zeno’s Max DAA	49
3.3.3	Zeno’s Parametric DAA	50
3.4	Mathematical Analysis of the Solutions	51
3.4.1	Bitcoin’s Default DAA	52
3.4.2	Zeno’s DAA	55
3.4.3	Zeno’s Max DAA	61
3.4.4	Zeno’s Parametric DAA	61
3.5	Discussion	65
3.5.1	Properties of Zeno’s Family	66
3.5.1.1	Discussion on P_1	66

3.5.1.2	Discussion on P_2	66
3.5.1.3	Discussion on P_3	67
3.5.1.4	Discussion on P_4	67
3.5.1.5	Discussion on P_5	68
3.5.2	Zeno's DAA vs. Zeno's Max DAA	68
3.5.3	Zeno's DAA vs. Zeno's Parametric DAA	69
3.5.4	Combining Zeno's Max and Zeno's Parametric	70
3.5.5	Zeno's DAA and Selfish Mining Minimum Profitability Hashrate	70
3.5.6	Implementing Zeno's DAA on Existing Blockchains	70
3.5.7	Impact of Zeno's DAA Family on Honest Miners	71
3.6	Summary	72
4	Evaluation and Results	73
4.1	Research Questions	73
4.2	Simulation Approach	74
4.3	Simulation Results	76
4.3.1	Selfish Mining	78
4.3.2	Profitability of Selfish Mining with Multiple Attackers	85
4.3.2.1	Two competitive selfish miners	87
4.3.2.2	Three selfish pools	90
4.3.2.3	Final Remarks on Selfish Mining with Multiple At- tackers	93
4.3.3	Zeno's DAA	95
4.3.3.1	Discussion on Q_1	95
4.3.3.2	Discussion on Q_2	98
4.3.3.3	The Effect Zeno's DAA on a Network without an Attacker	99
4.3.3.4	Zeno's DAA in Real World	101

4.3.4	Zeno’s Max DAA	104
4.3.5	Zeno’s Parametric DAA	108
4.4	Summary	110
5	Conclusion and Future Work	111
5.1	Future Works	114
	References	116
	Bibliography	125
A	The Bitcoin Simulator	126
A.1	The simulator components	127
A.1.1	The Block Class	127
A.1.2	The Event Class	127
A.1.3	The Miner Class	130
A.1.4	The Config Class	132
A.1.5	The Simulator Class	132
A.1.6	Other components	134
	Vita	

Chapter 1

Introduction

In this chapter, first, we briefly introduce and review the background necessary to understand the problem that we are trying to address in this thesis. Then, we discuss our motivations and contributions. In the end, we will present the structure of this thesis and talk about every chapter.

1.1 Problem Background

This section discusses key elements of the problem from bitcoin and how it works, to the difficulty adjustment process and the main focus of this thesis, selfish mining.

1.1.1 Overview of Bitcoin

Bitcoin, introduced in 2008 by Satoshi Nakamoto [54], is a digital currency, or in other words, as the author says, a distributed cash system. It is designed to remove the need for a trusted third party (e.g. Banks or Financial Institutions) for verifying the validity of financial transactions between two parties. In traditional models, a third party is needed to verify the validity of transactions. Instead, Bitcoin uses a distributed network of nodes (Bitcoin network), that helps to maintain the system. In fact, trust is not an issue here, and the integrity of the whole financial ecosystem

is guaranteed by design. Although it is not the first attempt to create such a system [16], it is the first complete and working solution to this problem.

For keeping track of transactions, Bitcoin uses a public ledger called ***blockchain***, which is a well-designed tamper-proof data structure. Every node in the network has its copy of this ledger and tries to validate it and add new transactions to it. The blockchain is a chain of blocks of data (in this case, Bitcoin transactions). Every block contains its parent (previous) block's hash. This makes them a chain and also gives the chain a sense of chronology. Because a hash of a block of data could not be computed before the block itself, every parent block is created before its child block. Moreover, with this structure, a block cannot be modified without altering all of its descendants (following blocks). When a block changes, its hash will change, so its child block (which contains the first block's hash) will change, so the hash of the child block will change, and this continues to the end. Having in mind that in Bitcoin, blocks keep transactions, it turns out that the transactions cannot be changed without changing its block's descendants.

To make the whole system secure, we need to have a method to make the block creation process hard, so that no one can generate many blocks easily in a short period of time. As it is called in Bitcoin, every node has to provide a ***Proof-of-Work*** (PoW) to show that it put a significant effort to create the block. So every newly generated block should meet a criterion. The criterion is that the hash of the newly created block should start with at least some (known number of) zeros. In other words, it should be lower than a particular number, usually referred to as *target* (note that every hash is, in essence, a number, often represented in binary or hexadecimal format). To do so, every node adds a random number, called *nonce*, to the end of the block's header and calculates its hash. If it fails to meet the criterion, the node will try with another nonce. But if it succeeds, the new block will be added to the blockchain, and the process starts over. Provided that the hash function is

a one-way function and cannot be reversed, the process of finding a valid nonce (or in other words, finding a Proof-of-Work) is a trial and error process, and there is no shortcut for it. In Bitcoin, Proof-of-Work is a puzzle that has no significance on its own other than proving that the miner put a notable amount of work into creating it. However, in some other blockchains like Primecoin [43], the proof-of-work actually solves a problem, in this case finding a pair of prime numbers. Note that not all blockchains use proof-of-work. Although proof-of-work is the most well-known and widely used type of consensus mechanism, there are other types of proofs e.g. *proof of stake*, *delegated proof of stake*, *proof of capacity*, etc. that are used in other types of blockchains.

The incentive for participating in this process is that every node who could successfully create a new block will receive some bitcoins as a reward. That is why this process is also called '*Bitcoin mining*' as it resembles mining new bitcoins with one's computational power. This way, every node participating in the process of mining, in addition to helping to keep the integrity of the system, earns some bitcoins.

1.1.2 Difficulty Adjustment

Bitcoin started in 2009 with a few personal computers as its network. Those PCs used their CPUs for calculating the hashes (mining process). Nowadays, there are millions of devices, mostly ASICs¹, each significantly stronger than those PCs. The number of hashes that the early Bitcoin network could generate in ten minutes could probably be generated by today's network in a matter of nanoseconds. This makes the blockchain unstable because one can generate many consecutive blocks in a very short period of time, even before the other miners receive its first block.

To solve this problem, Satoshi Nakamoto introduced a mechanism called '*difficulty*'. Before, we discussed that the criterion for a block to be accepted by other miners

¹Application Specific Integrated Circuit

is that the resulting hash of the block should start with a known number of zeros. Roughly speaking, this number is the difficulty of generating a new block (or in other words, calculating the PoW²). If this number is higher, it is more difficult to generate a PoW and vice versa. Because for example if the number of trailing zeros is n , on average it takes $\frac{2^n}{2}$ hash computations for the whole network to find one that begins with n zeros. So, we have a parameter that can be changed to make block generation easier or harder.

In Bitcoin, the goal is to keep the block generation rate constant. It means that we want the whole network to be able to generate a block every ten minutes on average. So, if the network grows, we have to increase the difficulty, and vice versa. In Bitcoin, this happens every 2016 block, which is roughly equivalent to two weeks. This process is called difficulty adjustment.

Bitcoin's Difficulty Adjustment Algorithm (DAA) is very simple: At the end of every 2016 blocks period, every node calculates the ratio of *expected block generation time* to the *actual block generation time*, then multiplies this ratio by the current difficulty value to get the next difficulty value. By doing so, they set the *difficulty value* of the next period to the actual *difficulty value* of the previous period³. Equation (1.1) shows Bitcoin's default difficulty adjustment algorithm. In this equation, D_{n-1} shows the difficulty value of the previous period, and D_n is the difficulty value for the next period. Because every node has the same blocks in their local version of the blockchain, even though they are calculating the next difficulty value separately, the results of their calculation would be exactly the same. Figure 1.1 shows the difficulty value for the Bitcoin network since the beginning, both in linear and logarithmic scale. We extracted the actual difficulty values for each period from Bitcoin's blockchain itself, which is publicly available.

²In the real world, it is a bit more complex and has more technical details, But the concept is the same.

³It is reasonable to assume that the difficulty value for the next period is equal to the actual difficulty value of the network that is calculated based on the data from the previous period.

$$D_n = D_{n-1} \times \frac{\text{last period's expected time}}{\text{last period's actual time}} \quad (1.1)$$

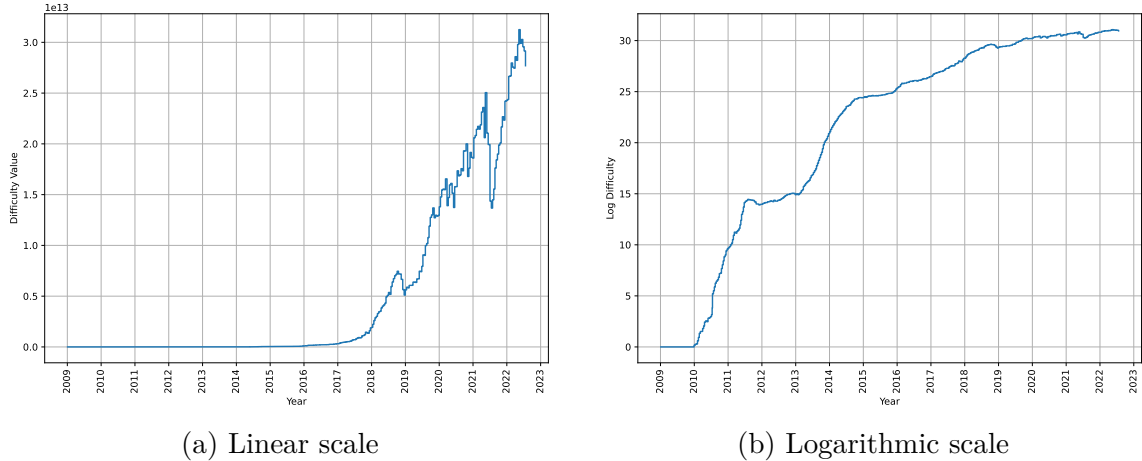


Figure 1.1: Bitcoin network difficulty from the beginning

1.1.3 Selfish Mining

Selfish Mining is an attack from a mining pool toward other miners or mining pools. Selfish miners do not follow the protocol when providing the proof-of-work. Instead, they follow a set of rules that help them waste honest miners' efforts, and therefore, increase their proportional reward. Briefly, when a selfish miner/pool finds a block, she will not propagate it immediately, but rather keep it for herself, so other miners cannot mine on the top of her block. Then she publishes the block whenever it is needed, to invalidate honest miners' blocks.

The set of rules that a selfish pool follows depends on the event of finding a block. Thus, it is divided into two subsets: 1) if the selfish miner found a block, and 2) if the honest miners found a block. In the case of each of those events and also based on the state of the network at the time of that event, the selfish mining strategy indicates what action the attacker should take. This action is different from the default behaviour that is expected from a miner. We will talk about this strategy

later in more detail. By following this set of rules, the attacker will waste a lot of resources from honest miners.

Eyal and Sirer analyzed this strategy, presented this process as a state machine, and computed the reward in every state and the final reward according to the probability of each state [21]. In their model, they have two parameters: The first one is α , which is the selfish pool mining power, relative to the computational power of the whole network. Thus, the honest miners' computational power would be $1 - \alpha$. The second one is γ which denotes the ratio of honest miners who choose to mine over selfish mining branch.

1.1.4 Importance of Selfish Mining

Selfish mining is introduced in an academic paper by Eyal and Sirer [21]. Since its introduction in 2013 until the present day, their paper has been cited over two thousand times, including papers that analyzed it in different situations, papers that extend it and combine it with other attacks, papers trying to detect and prevent it, etc. So, It is a very well-known problem in the blockchain world.

Note that because of the pseudonymity of the Bitcoin, an attacker could easily create a new identity (wallet address) for generating each block and avoid being detected. Because of this, selfish mining is a sneaky attack and is not easily detectable. There have been a few studies that attempt to detect selfish mining, for example, ForkDec [75]. Based on their analysis, no instances of the attack have been observed in Bitcoin's network yet. There might be a few reasons for that, one of them being since the introduction of selfish mining, there were rarely instances that a mining pool was strong enough to be able to perform selfish mining. However, this doesn't mean that in the future there will not be a strong pool tempted to try it [25].

Although selfish mining is dominantly addressed in the context of Bitcoin and our focus is also on Bitcoin itself, its impact is not limited to Bitcoin and it can be applied

to any other proof-of-work blockchain and cryptocurrency, e.g. Ethereum [57]. At the time, even with the recent drop in the price of cryptocurrencies, there are around fifty cryptocurrencies with a market capitalization of at least a billion dollars, many of which are proof-of-work blockchains and potentially susceptible to the attack. One of the instances in which selfish mining is observed in the real world was in 2018 on the first Japanese cryptocurrency called MonaCoin, where the attackers were able to create a private branch of nine blocks using selfish mining and cause around \$90,000 in damage [34]. This means that smaller altcoins are more vulnerable to this attack because it is easier to control a larger portion of hash power in them.

1.1.5 Blockchain Trilemma

Researchers have already discussed that the blockchain or essentially any distributed system can face a trilemma often known as the *scalability trilemma* [37,72]. The three components of this trilemma are:

- ***Decentralization***: It is the main focus of the blockchain that makes it possible to be controlled by a network of peers rather than an individual. It is arguably the easiest component to achieve.
- ***Scalability***: This refers to how well the network can be scaled to be able to process higher transaction throughput. This component one is the most challenging of the three components and is still considered one of the main issues in the blockchain that needs to be addressed.
- ***Security***: Without proper assurance of security, no one will trust to participate and use a distributed system. As such, the Blockchain should be resilient to external attacks and be able to defend its users.

Retaining all three of these components simultaneously is a challenging task, and improving one of them often comes at the cost of partially losing another. Our work

is also not an exception to this rule. In this work, we try to improve the security of the blockchain, especially against selfish mining attacks. This may lead to minor decline in scalability. We discuss this in more detail in the following sections.

1.2 Motivations and Contributions

Bitcoin and blockchains in general gained a lot of popularity over the last few years. With a price in the range of \$30K to \$65K from the start of 2021, the total market capitalization of Bitcoin varied between 600 billion dollars to more than one trillion dollars. Also, there are hundreds of cryptocurrencies other than bitcoin, many of which (fifty to a hundred of them) have a market capitalization of more than a billion dollars. In such an environment, the security of these systems and their mechanisms will be of great importance.

The scope of this research is to address one of the security issues of Bitcoin and other similar cryptocurrencies in the mining process. There are several attacks on the mining process of Bitcoin which will cover in the next chapter. The one that is the main focus of this research, as mentioned before, is *Selfish Mining*, which we investigate and try to find a solution for it.

- As our **first contribution**, we investigate the effectiveness of selfish mining and show that *time* is an essential factor in the profitability of selfish mining, and those attackers who are powerful enough can potentially benefit from selfish mining after a certain waiting period.
- For our **second contribution** we analyze cases with multiple selfish miners to show what would happen if there are multiple non-cooperative agents performing selfish mining in the system.
- We further argue that the difficulty adjustment algorithm in the Bitcoin network is a major factor for allowing selfish mining to work. Therefore, as our

third contribution, we propose an alternative difficulty adjustment algorithm, *Zeno's DAA*, that discourages selfish mining by making it less profitable and riskier for the attackers. To the best of our knowledge, our work is among the first to propose to adaptively adjust the difficulty. We show that it is possible to extend the waiting period needed for selfish mining to become profitable to more than double by using our proposed difficulty adjustment algorithm, which will discourage miners from performing the selfish mining attack.

- Our **forth contribution** would be extending *Zeno's DAA* to a family of difficulty adjustment algorithms that includes *Zeno's Max DAA* and *Zeno's Parametric DAA*, each of which has its unique characteristics and can improve *Zeno's DAA* in its own way.
- For our **fifth contribution**, we perform a mathematical analysis on our proposed family of difficulty adjustment algorithms to compare them with Bitcoin's default DAA and show their effectiveness.
- Finally, as our **sixth contribution**, we developed a Bitcoin network simulator. This simulator, which will be discussed in more detail in Section 4.2, is a discrete-event simulator that can be used to run arbitrary episodes of the Bitcoin network with different configurations of miners. It can simulate different mining strategies, in particular, default (honest) mining and selfish mining. Also, the simulator can be used to study the effect of various difficulty adjustment algorithms, namely default DAA and *Zeno's DAA*. The simulator could also be easily expanded to include more mining strategies and difficulty adjustment algorithms. So, we used this simulator to study and analyze selfish mining and the effectiveness of our proposed DAA.

1.3 Thesis Organization

The rest of the thesis is organised as follows:

- Chapter 2: *Background and Literature Review* provides an overview of Bitcoin system and some of its technical aspects. It also talks about Bitcoin threats including security and privacy issues, and previous works that has been done in this area. Then it talks about the main focus of the thesis, selfish mining, and present it with details.
- Chapter 3: *Methodology and Proposed Solutions* first, digs deep to answer a question: is selfish mining profitable or not? We talk about opposing views and the debate that exist around the topic. Then, we provide an answer and the big picture that help us better understand this attack and its profitability. We continue by explaining why there is a need for better difficulty adjustment algorithms and how it can help. Then we propose our solution to the problem, which is an alternative difficulty adjustment algorithm called Zeno's DAA, and we extend this algorithm to a family of DAAs by introducing two more DAAs, Zeno's Max DAA and Zeno's Parametric DAA, that improve Zeno's DAA in two different ways. After that, we provide a mathematical analysis of the solutions, and then conclude the chapter by discussing various point about alternative DAAs, from their properties and whether or not they have those properties, to applicability of them and what it takes to use them in the real world.
- Chapter 4: *Evaluation and Results* first talks about our evaluation methods: *mathematical analysis* and *simulation method*. After that it presents the mathematical analysis for Bitcoins's default DAA and also our proposed algorithms. Then, for the simulation method, we start with a case without selfish miner to check the simulator. Then we analyze cases with one and more selfish miners

to study their effect on each other and the network. And finally, we present the result of our simulations for our family of alternative DAAs and show their effectiveness.

- Chapter 5: *Conclusion and Future Work* concludes our work by presenting our findings, discussing our contributions and limitations, and suggesting future line of works for this research.

Chapter 2

Background and Literature Review

In this chapter, we review the literature and provide background information related to the goal of this thesis. We start with the principles of Bitcoin and how it works. Then we proceed with Blockchain and Bitcoin threats and discuss various threats and attacks to the privacy and security of the blockchain, with a focus on mining attacks and more specifically, selfish mining. Then we talk about a few mitigation techniques and their shortcomings.

2.1 Bitcoin Preliminaries

In this section, we are going to briefly explain how Bitcoin works. There are different aspects of the Bitcoin mechanism that needs to be explained in order to understand how the system actually works.

2.1.1 Identities

Identities in Bitcoin are associated with Bitcoin addresses, which are linked to a pair of ECDSA¹ public and private keys. Anyone can generate any number of Bitcoin addresses and use them. In the blockchain (which is the public ledger shared between

¹Elliptic Curve Digital Signature Algorithm

all nodes in the Bitcoin network), the only transaction will be stored. A transaction can have multiple inputs and multiple outputs. The input of a transaction is actually the output of a previous transaction. Whenever someone wants to transfer some money to another person, she will create a transaction with inputs that are unspent outputs of some previous transactions, and output that is the Bitcoin address of the second party, then sign it with her own private key linked with her Bitcoin address. Then broadcasts it to the Bitcoin network, so that nodes that act as *miners* can put it on the blockchain. The term “blockchain” comes from the fact that it is actually a series of blocks of data (in this case transactions) that are chained together using the hash. It means every block, in addition to the list of transactions, contains the hash of its previous block. This way, changing a block, even a single bit, makes every subsequent block invalid because the hash of the block will change.

2.1.2 Transactions

As we expect, in their simplest form, transactions represent sending a particular amount of money from one identity to another one. In the blockchain, there is no record of one’s account balance because it is too hard, or even impossible to keep account balances in sync, among multiple instances of the blockchain, which is a distributed ledger. Instead, there are just transactions that are stored on the blockchain. So, if you want to see the balance of an account, you should search for the transactions that send money to this account balance, and are not spent yet. Then you can compute their summation to get the account balance.

So everyone’s bitcoins come from previous transactions. Therefore, when someone wants to send some bitcoins to someone else, she should spend some previous transactions, and use them as inputs to the new transaction. Suppose Alice wants to send 3 bitcoins to Bob, and suppose Alice previously received 2 bitcoins from Charlie (in transaction TX1), and 2 other bitcoins from Dave (in transaction TX2). Then Alice

transactions in Bitcoin, there is a simple stack-based scripting language that could be used for creating more complex transactions. This scripting language is called *Bitcoin Script* or simple *Script*. Every transaction in Bitcoin (including previous examples) should also be written in Script, which is intentionally designed not to be Turing-complete, most importantly without loops. The reason for this is that every transaction should be validated by the miners, and having loops that open the possibility of having infinite loops, causes serious problems. Because if someone sends a transaction with an infinite loop, all the miners should validate it, but by doing so, they will get stuck in a loop and will never be able to validate the transaction. In its essence, a transaction is like a list of instructions that describes how the next owner of the coin can gain access to it and spend it. In the example above, we have a Bitcoin transferred to a destination address, say address D. If in the future, the user who owns the address D wants to spend her Bitcoin, she should provide two things:

1. A public key that is associated with the address D (It means when you hash the public key, you should get destination address D embedded in the script), and
2. A signature that proves she owns the private key corresponding to the public key.

By using this scripting capability, you can achieve the flexibility of defining how a particular amount of Bitcoin should be spent. In our typical examples, it just needs a set of public and private keys. But there are many more possibilities, for example, one can define a transaction that needs two sets of public keys to be spent, or even more sets of keys, or a password instead of keys, or even no keys at all. One interesting example is a particular transaction in Bitcoin that has been created in 2013, and whoever finds a hash collision in SHA1, can spend the output which was

worth 2.48 Bitcoin. In February 2017 someone found a hash collision and claimed the reward.

2.1.3 Bitcoin Mining

As discussed before, Bitcoin needs a group of distributed peers to keep its network running. These nodes are responsible for validating transactions and providing a Proof-of-Work to add the transactions as a new block to the blockchain. Providing the Proof-of-Work is a trial and error process and requires a lot of computational power. Therefore, there has to be an incentive for the peers to participate in this process.

To create motivation for people to join the Bitcoin network, the first transaction of a block is designed as a special transaction that creates new coins that are owned by the creator of the block. This is usually called *block reward*. This also provides a way to distribute coins into circulation. At the beginning of the Bitcoin network in 2009, the block reward was **₿50**, but after every 210,000 blocks, the block reward is cut in half to create a total amount of **₿21M** in the end. This process is called *halving*. The amount of block reward is **₿6.25** as of May 2020.

Another part of the reward is transaction fees. For every transaction that Bitcoin users create and send to the nodes to be included in the blockchain, the amount of output should be less than (or equal) to the amount of input. The difference between the amount of output and input is considered as the transaction fee. Nodes collect these transaction fees from the transactions that they include in a block by adding them to the block reward in the first transaction.

Since creating new blocks requires a lot of computational power and the result is receiving a reward in form of Bitcoins, this process resembles mining. Thus, it is called *Bitcoin mining*, and the nodes participating in it are called *Bitcoin miners*. Although finding a new block is completely random and determined by chance,

the amount of reward that each miner receives in a long run would naturally be proportional to its computational power.

The mining process itself has some vulnerabilities that people can use to attack it. The difference between an attack on the Bitcoin network and an attack on the mining process is that in the former, attackers try to manipulate transactions and steal bitcoins. So the target is Bitcoin users (meaning the people who own some bitcoins and try to make transactions with them). But in the latter, the target of the attack is miners themselves. Basically, in these types of attacks, miners try to perform the attack against other miners to increase their mining revenue with other miners losing some parts of their revenue. We will discuss these attacks in the following sections.

2.1.4 Pooled Mining

As the size of the Bitcoin network grows, its total computational power also increases and it becomes more difficult for individual miners to mine a new block. Finding a block has a significant reward, but the computational power of an individual is very limited in comparison to the computational power of the network, so an individual has a small chance of finding a block. Although, in other areas such as Edge Computing, the issue of resource-limited devices has been addressed by developing optimization algorithms to reduce the energy and delay cost [84, 85].

Because of the limited hashrate of a user, the variance of the reward is very high, because most of the time, a solo miner will receive no reward, but on a very rare occasion, she will receive a significant reward. To reduce income fluctuation, miners usually join forces and create pools to mine together and share the reward. With their computational powers combined, they will find blocks more often and divide the reward among themselves. By doing so, they will reduce the fluctuation of the reward significantly, because they will receive smaller rewards but frequently. Obviously, the

expected value of the reward for a single miner will not change, and she will receive the same reward in the long run.

2.2 Blockchain and Bitcoin Threats

There are different categories of threats and attacks against blockchains from different aspects, e.g. blockchain structure, peer-to-peer system, mining process, and applications [65] [66]. In a broader look, we can categorize Bitcoin threats into two major categories: *privacy issues* and *security issue*. There are multiple studies that survey these issues [14] [82] [53] [40] [33]. We will review these categories and a few examples of attacks related to them in the next sections.

2.2.1 Privacy and Anonymity Issues in Bitcoin

In traditional payment systems, the trusted third parties (e.g., banks) who are responsible for validating transactions, are also responsible for the privacy of the users. They achieve this goal by limiting access to transaction information. This information is only accessible by the parties involved and the trusted third party. However, the Bitcoin approach is totally different. The sole purpose of Bitcoin is to remove the need for a trusted third party and to achieve this goal, all the transactions are public and every node in the network has access to all the transactions. Transaction validation is done collectively by all the nodes. But this does not mean that there is no privacy in Bitcoin.

Identities of the users in Bitcoin are associated with Bitcoin addresses, which are linked to a pair of ECDSA² public and private keys. Anyone can generate any number of Bitcoin addresses and use them. This means every node in the network can see that someone (with a public address) is sending some bitcoins to someone

²Elliptic Curve Digital Signature Algorithm

else (also with a public address), but these addresses cannot be linked to anyone easily. Also, to further improve privacy, it is suggested strongly that users should use each address only once.

However, this is an ideal case and in the real world, linking public addresses to individuals is still possible to some extent [8]. Adversaries can perform blockchain analysis to follow the flow of the money in the Bitcoin network using some simple rules. For example, in multi-input transactions, all of the input addresses are almost certainly belong to a single individual [50], or the address that is provided to collect the change (after sending the specified amount to the receiver) is also belong to the sender. By using these rules, adversaries can link addresses together, and using other off-chain information (for example public addresses that a Bitcoin user publishes in their website or forums or even in emails) they can link this group of addresses to a single individual or IP address [44] [24] [60]. Androulaki et al. performed an experiment to deanonymize users in such networks [2] and it showed promising results.

Therefore it is often stated that Bitcoin does not offer anonymity. Instead, it offer *pseudonymity*. This problem is not limited to Bitcoin. Many other blockchains and cryptocurrency systems suffer from the same problem [42].

Privacy Improvement Proposals

There have been various efforts to mitigate privacy issues in Bitcoin [9] [7] [23]. They can be categorised as three major categories: *Peer-to-peer mixing protocols*, *Distributed mixing networks*, and *Altcoins* [14].

- **Peer-to-peer mixing protocols:** Mixing is a technique that can be used to confuse the trail of transactions to remove linkability in Bitcoin and preserve privacy. In peer-to-peer mixing, mixers who are anonymous service providers receive Bitcoins that are divided into small parts from different users, mix them

into a transaction and send them back to the users but with different public addresses. This helps break links between public addresses and ensures the privacy of the users. There are many examples of these mixing protocols, e.g. CoinJoin [49], CoinShuffle [63], Xim [10], etc.

- **Distributed mixing networks:** MixCoin [11] is one of the examples in this category. Users share some bitcoins with MixCoin, which is a third party and receives back the same amount of coins. So it provides strong anonymity. However, there is the issue of trusting the third party, who could steal the coins or reveal sensitive information. To solve this problem, other mixing networks have been proposed. For example, BlindCoin [71] which uses blind signatures to create user inputs and cryptographically blinded outputs called blinded tokens. Another example is TumbleBit [35] which helps users to make fast, off-blockchain payments anonymously through an untrusted intermediary called Tumbler.
- **Altcoins:** Bitcoin is the first and most famous cryptocurrency in the world. But there are tens of other coins that are inspired by Bitcoins. These coins are called *altcoins*. Some of these altcoins try to tackle privacy issues in Bitcoin and offer different mechanisms to ensure the anonymity of their users. One of these altcoins is ZeroCoin [52] that uses cryptographic techniques such as zero-knowledge proof to and can validate encrypted transactions. Two extensions of ZeroCoin are ZeroCash [68] and Zcash [38], which try to further develop ZeroCoin and overcome some of its limitations, using advanced cryptographic techniques such as zk-SNARK. Other examples include *Monero* which is based on CryptoNote protocol and uses ring signatures to improve the privacy of its users, and *Dash* [19] which mixes transactions using master nodes to achieve higher privacy.

2.2.2 Security of Bitcoin

In this thesis, our focus is on attacks against the mining process or in other words, the peer-to-peer structure of the blockchain. Many of these attacks are related to the pooled mining process that we discussed before.

There are several categories of attacks against Bitcoin and other PoW blockchains in general [29] [15] [69]. One of these categories is mining attacks. Attacks in this category target the mining process in Bitcoin. These attacks are usually not related to the security of the Bitcoin and Bitcoin network, meaning they are not trying to compromise the validity of the blockchain (the distributed ledger) or steal bitcoins. Instead, they target the miners and pools. In a scenario where all miners are working honestly (loyal to Bitcoin’s protocol), in the long run, any miner will receive a reward proportional to her computational power³. For example, let’s suppose that the total computational power of the entire Bitcoin network is one. Then suppose one miner or a pool of miners has a fraction of it, say α . So, the ratio of the reward for this miner should also be α . We call it the miner’s ‘*fair share*’ of the rewards. In mining attacks, attackers try to increase their revenue beyond their fair share and/or decrease other miners’ (or pools) revenues. We discuss some of these attacks here.

Block withholding attack, introduced by Rosenfeld [62] is an attack against a mining pool, in which, the attacker joins a pool and pretends to contribute to the pool without any actual contribution, and receives her share from the pool, without any benefit for the pool [4] [5].

Another network-level attack, called the Eclipse attack, is a well-known attack that tries to isolate some parts of the network from the rest of it to be able to deceive them with an unreal picture of the network. Heilman et al. [36] introduced this attack in 2015.

One of the most famous attacks in this category is ‘*Selfish Mining*’, introduced by

³Terms ‘*computational power*’ and ‘*hashrate*’ both refer to the number of hashes that a miner generated per unit of time. We use these two terms interchangeably throughout this thesis.

Eyal and Sirer [20] in 2013. In selfish mining, the attacker intentionally creates forks. What it means is that in this attack, the attacker (also known as the selfish miner) starts to mine, and if she finds a new block, unlike what the protocol states, she does not publish its block to other miners. Instead, she keeps it for herself and tries to find the next block on top of it. However, if another miner finds a block, she will publish her block to invalidate the other miner's block. By doing so, she can waste the computational power of other miners in the network, so she will relatively gain more revenue than her fair share.

The importance of selfish mining is that it is not limited to Bitcoin, and it can be applied in other blockchains, e.g. Ethereum [22] [32] [61]. Based on [73] [46], until now pools in the Bitcoin network have not been large enough for selfish mining to be profitable, and the only case of selfish mining in the real world has been observed in Monacoin [64]. Although it does not guarantee that this will not happen in the future. There has been a few studies that analyze this attack [27] [56] [41]. We will discuss this attack in detail later on. Many other attacks in this category try to expand or generalize selfish mining attack [48] or combine it with other attacks [6] [18]. Here is a list of the most famous variants of selfish mining:

1. ***Stubborn Mining***: Presented by Nayak et al. [55] in 2015, this attack tries to improve selfish mining by using a new strategy. As the name suggests, in this strategy the attacker acts more stubbornly and keeps mining on her private branch even though it is not the longest branch. They also combine it with the Eclipse attacks, introduced before, to make it even more profitable.
2. ***Optimal Selfish Mining***: In this attack, Sapirshtein et al. (2016) [67] provide an algorithm to find an optimal selfish mining policy that guarantees a lower bound for the revenue.
3. ***Fork After Withholding Attack***: Introduced by Kwon et al. [45] in 2017,

this attack in some ways combines Selfish Mining and Block Withholding attacks to overcome their limitations and create a stronger attack.

Finding a solution to prevent selfish mining, which is the ultimate goal of this work, can potentially mitigate other similar attacks that use a similar mechanism.

2.3 Selfish Mining

Selfish Mining, introduced by Eyal and Sirer in 2013 [21], as discussed before, is an attack from a mining pool toward other miners or mining pools. This attack is designed specifically for Proof-of-Work blockchains and takes advantage of the characteristics of this type of consensus: generating a new block using the trial and error process, and sharing the newly generated block with the other miners. Therefore, it is not applicable in other consensus mechanisms like proof of stake. Selfish miners do not follow the protocol. Instead, they follow another set of rules, and by doing so, they waste honest miners' hashpower and increase their proportional reward. The main idea is that when a selfish miner generates a block, she will keep it private rather than propagating it to the network. So, other miners do not have access to her block (which is the last block in the blockchain) and therefore cannot mine on its top. Then the selfish miner will publish the block whenever necessary (according to a specific set of rules) and invalidate honest miners' blocks.

The set of rules that a selfish pool follows depends on the event of finding a block. Thus, it is divided into two subsets: 1) if the selfish miner found a block, and 2) if the honest miners found a block. This is the set of rules that one should follow if she wants to perform selfish mining:

- If she (the selfish miner) finds a block:
 1. If before finding this block, there was a tie (length of the public branch

and the private branch were equal), she is now ahead of honest/other miners. So, she publishes the entire private branch and will win the race.

2. Otherwise (meaning that she is way ahead⁴), she continues mining on her private branch.

- If others (honest miners) find a block:

1. If they are ahead, it means they win the race. In such a case, the attacker switches to their branch immediately.

2. In the case of a tie, she reveals her private branch immediately, hoping she will win the race.

3. If she is one block ahead, she reveals her entire branch and will win because she is ahead of them.

4. Otherwise (meaning she is way ahead of them), she reveals her first unpublished block and continues mining on her private branch.

By following this set of rules, the attacker will waste a lot of resources from honest miners. Eyal and Sirer analyzed this strategy and presented it as a state machine (See Figure 2.2). This diagram also shows the transition probabilities between the different states. Using this diagram, they computed the state probabilities as follows [21, Eq. (2)-(5)]:

$$p_1 = \frac{\alpha - 2\alpha^2}{2\alpha^3 - 4\alpha^2 + 1} \tag{2.1}$$

$$p_0 = \frac{1}{\alpha} \times p_1 \tag{2.2}$$

$$p'_0 = (1 - \alpha) \times p_1 \tag{2.3}$$

$$\forall k \geq 2 : p_k = \left(\frac{\alpha}{1 - \alpha}\right)^{k-1} \times p_1 \tag{2.4}$$

⁴Note that because of this strategy, the selfish miner is never behind the honest miners. Because if it happens, the selfish miner immediately switches to honest miners' branch

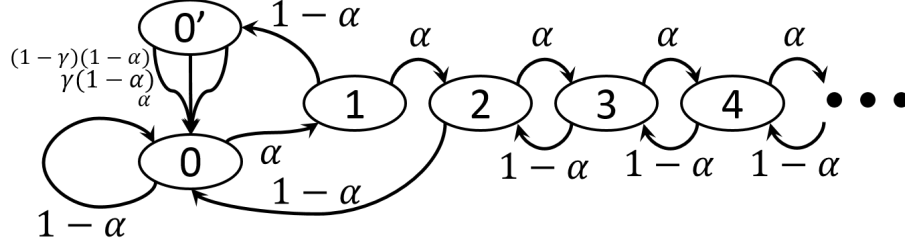


Figure 2.2: Selfish mining state machine with transition frequencies [21, Fig. 1]

in which, α is the selfish pool's hashrate, relative to the hashrate of the whole network. Consequently, the hashrate of all the other miners combined is $1 - \alpha$. In the equations above, p_1 shows the probability of being in state '1'. Likewise, p_0 , $p_{0'}$ and p_k show the state probabilities for states '0', '0'', and 'k' (for $k \geq 2$) respectively.

In their model, they also use another parameter, γ , which denotes the ratio of honest miners who choose to mine over selfish mining branches. This parameter does not affect the state probabilities, but it will show its effect on the revenue calculations. The next step is to calculate the revenues of the selfish pool and honest miners. If the selfish pool would have been honest, the ratio of its share from the total revenue would be proportional to its hashrate, α . However, in this scenario, we should calculate its revenue and also the revenue of honest miners when the pool performs selfish mining. The revenue of a miner is the sum of products of state probabilities and state revenue, over all the states. Based on different cases that might happen in every state, Eyal and Sirer calculated the two equations below (Equations 6 and 7 from [21]) that show revenues of the pool and others as:

$$r_{others} = p_{0'} \cdot \gamma(1 - \alpha) \cdot 1 + p_{0'} \cdot (1 - \gamma)(1 - \alpha) \cdot 2 + p_0 \cdot (1 - \alpha) \cdot 1 \quad (2.5)$$

$$r_{pool} = p_{0'} \cdot \alpha \cdot 2 + p_{0'} \cdot \gamma(1 - \alpha) \cdot 1 + p_2 \cdot (1 - \alpha) \cdot 2 + P[i > 2](1 - \alpha) \cdot 1 \quad (2.6)$$

As noted before, α , which is the hashrate of the selfish pool, also shows the ratio of

the pool's revenue in the case of honest mining.

To calculate the relative revenue of the selfish miner, they simply divided the value of r_{pool} by the sum of r_{pool} and r_{others} to get what they call 'revenue rate ratio' [21, Eq. 8]:

$$R_{pool} = \frac{r_{pool}}{r_{pool} + r_{total}} = \dots = \frac{\alpha(1 - \alpha)^2(4\alpha + \gamma(1 - 2\alpha)) - \alpha^3}{1 - \alpha(1 + (2 - \alpha)\alpha)} \quad (2.7)$$

Then they plotted the value of R_{pool} for $0 < \alpha < 1/2$ and different values of γ [21, Fig. 2], and also backed it up with a simulation analysis (also showed in the plot).

You can see this plot in Figure 2.3.

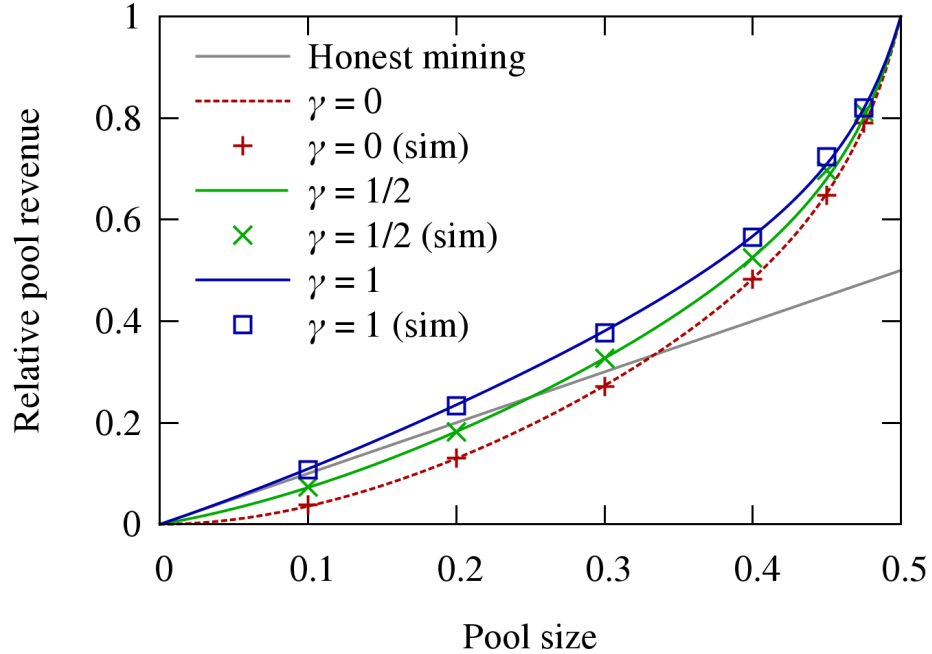


Figure 2.3: Relative pool revenue for a selfish miner [21, Fig. 2]

This figure shows that there are many cases that the pool revenue is more than alpha, which means the pool gains a revenue more than its fair share, using selfish strategy.

The point of profitability of selfish strategy is the point that R_{pool} becomes more than α . It means that if R_{pool} is more than α , then the selfish strategy is profitable for the pool. That is because in the honest case, the relative revenue of the pool

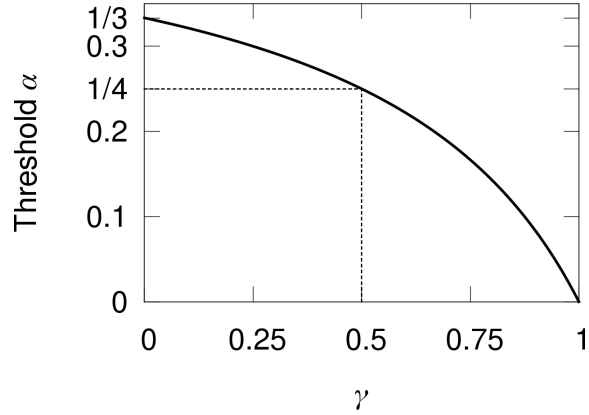


Figure 2.4: The minimum hashrate required for performing a profitable selfish mining attack as a function of γ [21, Fig. 3]

would be equal to its relative mining power, which is α . Therefore, by solving the inequation form Equation (2.8), Eyal and Sirer came up with a threshold for α with respect to γ .

$$R_{pool} = \frac{r_{pool}}{r_{pool} + r_{total}} > \alpha \Rightarrow \frac{1 - \gamma}{3 - 2\gamma} < \alpha \quad (2.8)$$

This inequation is a threshold for α . If α is more than this threshold, the selfish mining will be profitable for the pool, and its revenue will increase if they follow the selfish mining strategy. As can be seen here, it is dependent on γ , and it makes sense because γ plays a role in the profitability of selfish mining for a pool. Because if more honest miners accept selfish pool blocks, it would be more profitable for the selfish pool to mine selfishly than the case that honest miners do not accept their block. Figure 2.4 shows this threshold which is the minimum hashrate that a miner needs so that she can gain more revenue than her fair share if she perform the selfish mining attack.

2.3.1 Detection and Mitigation

Because of the decentralized nature and anonymity (pseudonymity) of Bitcoin and blockchain, detecting selfish mining is not so straightforward. However, there have been some attempts to detect this attack [13] [76]. For instance, one method of detecting selfish mining proposed by Saad et al. [64]. They leveraged the expected transaction confirmation height and the block publishing height to detect selfish mining behaviour in PoW-based Blockchains. Using the relationship between the two features, they created a “truth state” for each published block in order to distinguish between a legitimate block and a selfishly mined block.

There are some mitigation methods proposed to prevent selfish mining [56]. The first mitigation strategy was proposed by Eyal and Sirer [21], which suggests a slight change in the Bitcoin protocol. Bitcoin protocol requires that if you receive two blocks with the same heights, you as a node should accept the first one, broadcast it to others, and ignore the second block completely. In this mitigation, Eyal and Sirer suggest that in this situation, you should broadcast both of those blocks, and choose one of them *randomly*. They prove that by using this technique, you will set the minimum hash power required by a selfish miner to 25%. So, it is called a 25% defence against selfish mining, because by using this technique, the selfish miner should have at least 25% of the network hash power to benefit from selfish mining.

Another mitigation technique is called *Freshness Preferred*, proposed by Heilman [35]. In this method, every block should have a timestamp. These timestamps are unforgeable (meaning you cannot set a timestamp in the future) because every minute one timestamp will be generated and broadcasted to the network. Every miner should include this timestamp in their block, and whenever a miner receives a block with the same height, it will compare the timestamps of the newly received block and the previous block with the same height together and it (as the method name suggests) prefers the fresher one. Because, if a selfish pool withholds a block

it has been found, its timestamp will not be fresh. Heilman analyzed this method and proved that this method is actually a 32% defence mechanism, meaning every pool that wants to perform selfish mining profitably, should possess at least 32% of the computational power of the whole network.

There is also another mitigation technique proposed by Solat and Potop-Butucaru, called Zeroblock [70]. This method also suggests some changes to the Bitcoin protocol and introduces a new type of block besides regular transaction blocks, called Zeroblock. Zeroblock is essentially a dummy block with no significance in terms of data. The technique is as follows: Every miner is trying to mine a block, and also is listening to other blocks if there was a new block generated by other miners. If none of those happen in a fixed time window (for example 10 minutes, which is average block generation time), she generates a Zeroblock (which does not need a PoW, is easy to generate and can be created by all nodes independently), and appends it to its local version of blockchain. From now on, this block is the new head of the blockchain and all miners should mine on the top of this block, instead of the previous block. So if in the future, she receives a withheld block, she will know that this is withheld because it is not mined on the top of the Zeroblock. This defence is demonstrated in Figure 2.5. The authors analyzed this method and proved that it is a 50%+ defence mechanism. It means that selfish mining is not profitable unless the selfish pool has the majority of computational power, and of course, if it has, the original 50%+ attack will happen on the Bitcoin network and it will devalue the whole network.

There are a few problems with these mitigation techniques. More specifically, in freshness preferred we need an unforgeable timestamp, and so an entity that generates these timestamps. First, having such an entity challenged the distributed nature of the blockchain. Second, achieving such an entity is extremely difficult, and maintaining its security is another issue. Regarding the method Zeroblock, by

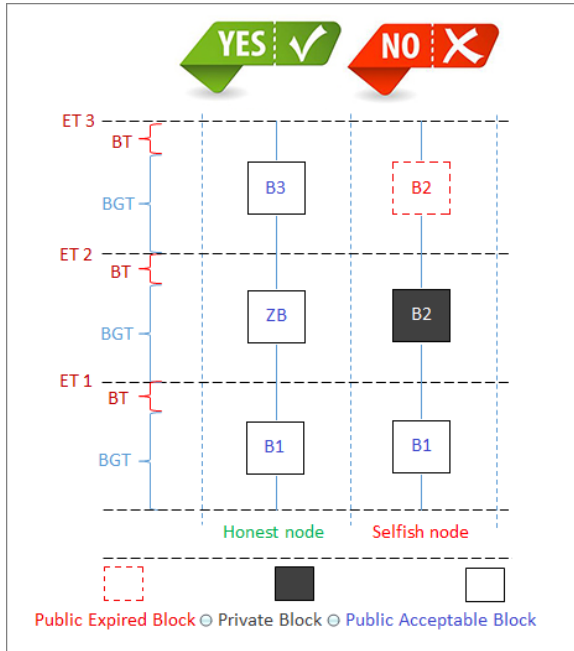


Figure 2.5: Honest miners generate a dummy block called *Zeroblock* to prevent selfish miners from performing their attack. This will invalidate any withheld block that is published after the time window of ET, which is the *expected time* for receiving or generating a new block. BGT and BT are expected value for *block generation time* and *block propagation time* respectively. [70, Fig. 1]

adding a dummy block to the blockchain, because this dummy block cannot include any transactions, we have periods that no transaction can be confirmed. This means transaction confirmation time for those transactions will increase by at least the time of the zeroblock. One other problem with these mitigation techniques is that they are fairly complicated and need significant changes to be made to the Bitcoin protocol. Because of this, the Bitcoin community has not adopted any of them as a solution to be added to the Bitcoin protocol. Also, these solutions do not pay attention to the difficulty adjustment algorithm, which is the reason that selfish mining becomes profitable. We will discuss this in the following sections. Our proposed solutions should overcome these issues and be practical to be used in the Bitcoin protocol.

2.4 Summary

In this chapter, we explained the principles of Bitcoin and how it works. Then we reviewed the privacy and security threats related to Bitcoin and blockchain in general. Then we discussed selfish mining which is the main focus of this thesis. After

that, we reviewed a few mitigation techniques and discussed their shortcomings. In the following chapters, we talk about our approach to the problem and our proposed methods.

Chapter 3

Methodology and Proposed Solutions

In this chapter, first, we investigate the debate around the profitability of selfish mining, settle the debate and depict a big picture to better understand the situation. Then, we argue why we need alternative difficulty adjustment algorithms and why they help. After that, we talk about the properties that a good alternative DAA should have. Then we proceed with proposing our proposed DAA, Zeno's DAA, and extending it to a family of DAAs with two more members: Zeno's Max DAA and Zeno's Parametric DAA that improve Zeno's DAA in two different ways. Then, we provide a mathematical analysis of the solutions. In the end, we discuss a few points about various aspects of the solution from satisfying the properties to applicability.

3.1 Profitability of Selfish Mining

In this section, we review countering arguments on the profitability of selfish mining and discuss their points. Then we introduce profitability metrics that will help us analyze the attack and compare the results. After that, we provide the big picture to show how both sides of the argument are partially right but inconclusive. Then

we talk about the effect of difficulty adjustment and also time on the revenue of the selfish miner and profitability of the attack.

3.1.1 Empirical Evidence

There has been several studies that analyze selfish mining from different perspectives, e.g. selfish mining with presence of propagation delay in the network [30] [74] [47] [83], multiple selfish miners [3] [80], selfish mining in imperfect networks [81] [39] As discussed in the previous chapter, Eyal and Sirer who first discovered and introduces selfish mining attack, analyzed selfish mining and presented it as a state machine with the transition probabilities between the different states in Figure 2.2. Using this state machine, they computed the state probabilities in Equations (2.1) to (2.4) as functions of the hash rate α . Based on different cases that might happen in every state, Eyal and Sirer calculated the revenues in Equations (2.5) and (2.6) and called them r_{others} and r_{pool} , which show the revenues of honest miners and the selfish pool, respectively.

The next step in Eyal and Sirer's calculation was simply calculating the relative revenue of the pool by dividing r_{pool} to sum of r_{pool} and r_{others} in [21, Eq.8] to get R_{pool} . Then, they demonstrated this value in [21, Fig.2] and showed that in many cases, the value of R_{pool} exceeds α , which means selfish mining is profitable for the selfish miner.

However, there is another side to this argument. Craig Wright is one of the most famous and influential people in Bitcoin community. He openly claims that he is Satoshi Nakamoto [77], the creator of Bitcoin, and of course many people believe him. He published two papers in 2017 to critique the selfish mining [79] [78]. In these papers, he claims that selfish mining is a fallacy and it is not profitable under any circumstances. He presents an analysis and demonstrate the results in [78, Fig.8]. This figure can be seen in Figure 3.1. In this figure, it is shown that the revenue

of a selfish miner (the red curve) never exceeds the value of α from honest mining (the black line). This contradicts the analysis of Eyal and Sirer which was presented in [21, Fig.2].



Figure 3.1: “Sketch of expected pool revenue for selfish-mine strategy.” [78, Fig. 8]

This is the debate that exist around the subject of profitability of selfish mining. Now, the question is that which side of this argument is correct? In the following sections, we take a deeper look at the profitability analyses and settle the argument.

3.1.2 Profitability Metrics

To be able to analyze the profitability, we need some evaluation metrics. The simplest, yet most important, evaluation metric is *total net revenue*. Because it is the most important, and very likely the sole factor for miners to participate in Bitcoin mining. The reason for emphasizing the term *total net revenue* is that, as discussed before, there are some cases in which, the *relative revenue* of the miners will increase. But that is not desired, because a miner does not want her total net revenue decrease, even if her relative revenue increases.

As an example, suppose Alice and Bob both make five bitcoins per day. Also, assume there is a strategy (e.g. selfish mining) that Alice can adopt. If she does this, Bob's revenue will be decreased to two bitcoins per day, but her revenue will also be reduced to four bitcoins per day. In this example, Alice's relative revenue increased in comparison to Bob's, and now she makes twice as Bob. But her total net revenue decreased by one bitcoin per day which is not desirable for a rational agent. Note that the total revenue of all miners (Alice and Bob) in this example is reduced from ten to six. This example shows that the total net revenue is an important evaluation metric, and in fact it is more important than relative revenue. Since the number of bitcoins mined is directly related to the number of blocks generated (currently 6.25 bitcoins per block), we only consider the number of blocks generated by a miner as her total net revenue and denote it as R . Suppose t is the time (in hours) since a particular miner P_i started to mine.

$$R_{P_i}(t) : \quad \begin{array}{l} \text{Number of blocks generated by the pool } P_i \text{ in time } t \\ \text{(Total net revenue of the pool } P_i) \end{array} \quad (3.1)$$

The scenario where all miners are honest is our baseline, which defines our expected value of the revenue. Because in this scenario, every miner will receive its fair share of the total revenue, and any deviation from this amount shows either loss or profit. We denote the revenue in the all-honest scenario as \bar{R} as below.

$$\bar{R}_{P_i}(t) = \alpha_i \times t \times 6 \quad (3.2)$$

The number six in the equation above refers to the fact that on average, six blocks should be generated by the whole network every hour. In other words, the block generation rate for Bitcoin is one block every ten minutes.

For every miner, if all miners are loyal to the protocol (mine honestly), their expected total net revenues are the same as \bar{R} and proportional to their hashrate, α_i .

$$R_{P_i}(t) \approx \bar{R}_{P_i}(t), \text{ if all miners are honest} \quad (3.3)$$

By comparing R_{P_i} and \bar{R}_{P_i} in the presence of selfish miners, we can see if the configuration of miners has any profit or loss for every miner or not. Therefore we define another metric to see this comparison. It is a normalized gain or loss metric which we call it ‘*gain*’ for short, denote it by G_{P_i} and define it as follows:

$$G_{P_i}(t) = \frac{R_{P_i}(t) - \bar{R}_{P_i}(t)}{t \times 6} \quad (3.4)$$

In the case of all miners being honest, G_{P_i} should be zero. Because the numerator of the equation above equals zero (as $R_{P_i}(t) \approx \bar{R}_{P_i}(t)$).

3.1.3 Findings

An important point that is not mentioned in either analysis is difficulty adjustment and how it affects profitability. In this section, we revisit the analyses by taking the difficulty adjustment and time into account.

As noted before, α , which is the hashrate of the selfish pool, also shows the ratio of the pool’s revenue in the case of honest mining. In Eyal and Sirer’s calculations, r_{pool} [21, Eq. 7] shows the new ratio of the pool’s revenue in the case of selfish mining. Therefore, we can interpret this ratio as ‘*effective hashrate*’ (or ‘*apparent hashrate*’ [31]) of the pool in this scenario. Note that before the difficulty adjustment, the effective hashrate of the pool is lower than α .

$$\forall \alpha \forall \gamma : r_{pool} < \alpha$$

Figure 3.2 shows the amount of r_{pool} which is the total net revenue (not relative

revenue) of the selfish pool with respect to α for different values of γ . As it is shown in this figure, the value of r_{pool} is always below the black line, which corresponds to the revenue of the pool if it chooses the honest mining strategy. This is also the case for other honest miners in the network, meaning they also lose some of their revenue and their effective hashrate is lower than $1 - \alpha$.

$$\forall \alpha \forall \gamma : r_{others} < 1 - \alpha$$

As a result, the total revenue in the network is lower than one. This indicates that with the presence of selfish miners, there will be many discarded blocks in the network, which leads to a lot of waste in computational power. Because of this, the total revenue that is generated in the network will decrease. As it is shown below, r_{total} which is the sum of revenues of the pool and other honest miners, is always

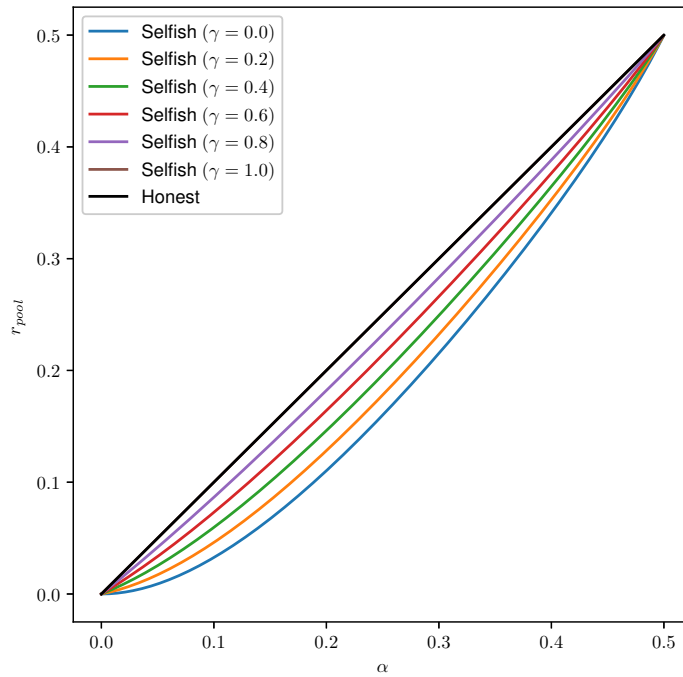


Figure 3.2: r_{pool} for different α and γ (after difficulty adjustment)

lower than one. It means that from the perspective of the total net revenue, selfish mining is never profitable.

$$r_{total} = r_{pool} + r_{others} < 1 \quad (3.5)$$

If there is no difficulty adjustment in the system, this is the case. When a miner starts to use the selfish mining strategy, all the nodes in the network will suffer a loss. However, the loss may be less for the selfish miner in some cases. The analysis above was in the case in which there are no difficulty adjustments (like the period between two difficulty adjustments). This is what Wright and Savannah referred to in their paper [78].

They argue that this scenario will always hold. However, in practice, the Bitcoin network operates using a difficulty adjustment algorithm. What will happen in the long term with the presence of difficulty adjustments? To calculate the relative revenue of the selfish pool, Eyal and Sirer simply divided the selfish pool revenue (Equation (2.6)) by r_{total} (Equation (3.5)), the sum of the revenues of both honest miners (Equation (2.5)) and selfish pool, and called it R_{pool} . As noted before, α , which is the hashrate of the selfish pool, also shows the ratio of the pool's revenue in the case of honest mining. However, r_{pool} shows the new ratio of the pool's revenue in the case of selfish mining. Therefore, we can interpret this ratio (R_{pool}) as '*effective hashrate*' after difficulty adjustment. The result is [21, Eq. 8]:

$$R_{pool} = \frac{r_{pool}}{r_{pool} + r_{others}} = \frac{r_{pool}}{r_{total}} \quad (3.6)$$

Figure 3.3 shows the value of R_{pool} with the same configuration as Figure 3.2. It shows that in many cases the revenue of the pool exceeds its revenue in case it stays honest. This is a strong motivation to perform this attack against other miners in the network. The fact that the difficulty adjustment makes the selfish mining profitable,

is the most important point in this thesis and we will get back to it in the following sections.

An important factor that neither Eyal and Sirer [21] nor Wright and Savanah [78] considered in their analyses is *time*. In the next section we discuss the effect of time on profitability of selfish mining.

3.1.4 Profitability of Selfish Mining in Time

As we discussed before, Eyal and Sirer considered the final value of the revenue many periods from the start of the attack, while Wright considered just the first period of the attack and before any difficulty adjustment. In a sense, they are both right. However, to better understand the revenue and its changes, the important time factor has to be considered.

Based on the analysis in the previous section, regardless of the selfish miner's hash

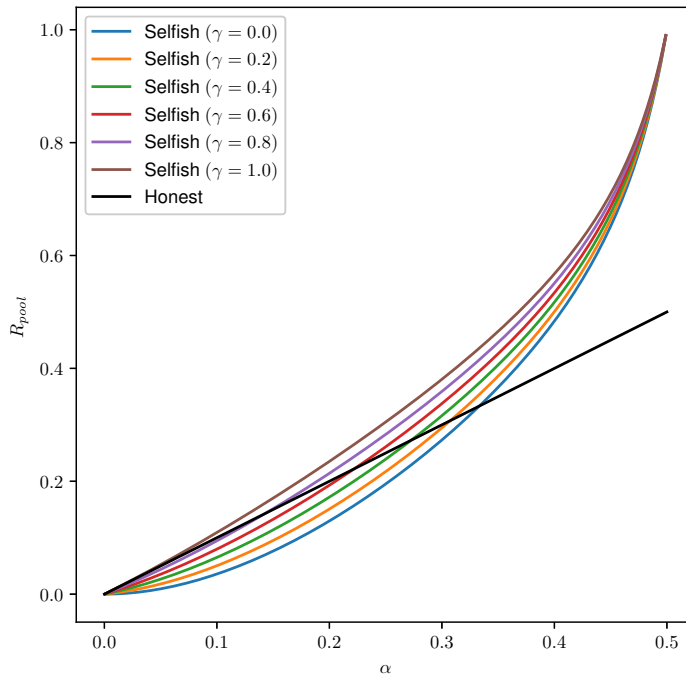


Figure 3.3: R_{pool} for different α and γ (after difficulty adjustment)

power, selfish mining is not profitable at the beginning. However, after difficulty adjustment, the revenue of the selfish miner starts to increase. If the selfish miner is powerful enough, it will eventually exceed the honest mining strategy's revenue.

Here we want to analyze the effect on time on the revenue of a selfish miner with the presence of difficulty adjustment. For the sake of simplicity, we assume that the time begins from the moment that a pool starts to adapt the selfish mining strategy right after a difficulty adjustment (beginning of a new period). Also, we mentioned that in Bitcoin, the difficulty adjustment takes place every 2016 blocks, and the block generation rate is one block every ten minutes or $\frac{1}{6}$ hours. So the length of a difficulty adjustment periods should be as follow:

$$T = 2016 \times \frac{1}{6} = 336 \text{ hours} \quad (3.7)$$

Suppose that at the beginning, the value of difficulty equals one. When a pool starts to perform the selfish mining strategy, for the first period, the value of difficulty stays equal to one. Meanwhile, selfish mining leads to a lot of discarded blocks, which leads to lower block generation rate in the first period, before the difficulty adjustment tries to bring it back to its intended value of six blocks per hour. Therefore, the block generation rate in the first period will drop by a factor of r_{total} , which is the effective hashrate of the entire network in the first period. As a result, the length of the first difficulty period, T_1 , will be longer than T and it has a reverse relationship with r_{total} .

$$T_1 = \frac{T}{r_{total}} > T \quad (3.8)$$

After 2016 blocks, the difficulty adjustment happens. We assume that after T_1 , nothing will change, meaning that no miner will change its strategy, no miner will

leave the network, and no new miner will join the network.

$$D_0 = 1 \text{ (initial difficulty)}$$

$$D_1 = D_0 \times r_{total}$$

$$D_{k>1} = D_1$$

By converting the difficulty to a function of time, we will have:

$$D(t) = \begin{cases} 1 & , t < T_1 \\ r_{total} & , t \geq T_1 \end{cases} \quad (3.9)$$

Figure 3.4 shows the value of difficulty with respect to time.

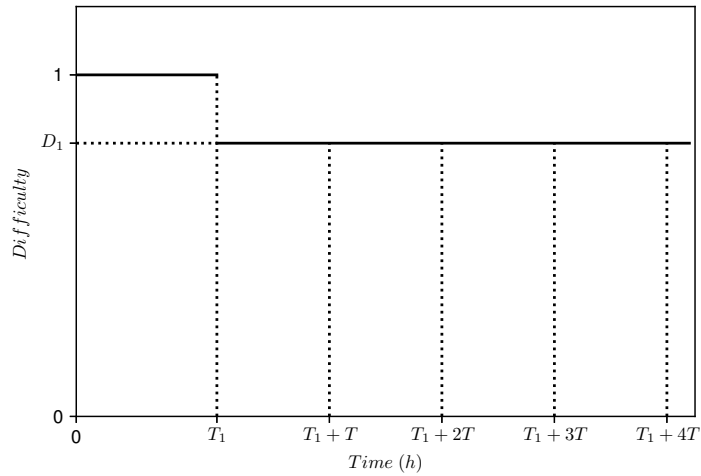


Figure 3.4: Difficulty value by time in a selfish mining scenario

Now, we calculate Equation (3.6), but this time we consider the time and this means

we have to consider the effect of the difficulty adjustment:

$$\begin{aligned}
R_{pool}(t) &= \int_0^t \left(\frac{r_{pool}}{D(t)} \times 6 \right) dt \\
&\text{if } t > T_1 \implies \\
R_{pool}(t) &= \int_0^{T_1} \left(\frac{r_{pool}}{1} \times 6 \right) dt + \int_{T_1}^t \left(\frac{r_{pool}}{r_{total}} \times 6 \right) dt \\
&= \left[\frac{r_{pool}}{1} \times 6t \right]_0^{T_1} + \left[\frac{r_{pool}}{r_{total}} \times 6t \right]_{T_1}^t \\
&= r_{pool} \times 6T_1 + \frac{r_{pool}}{r_{total}} \times 6(t - T_1) \\
&= \frac{r_{pool}}{r_{total}} \times 6t - \left(\frac{r_{pool}}{r_{total}} - r_{pool} \right) \times 6T_1 \\
&= R_{pool} \times 6t - (R_{pool} - r_{pool}) \times 6T_1 \tag{3.10}
\end{aligned}$$

In Equation (3.10), as seen previously, r_{pool} and R_{pool} show the effective hashrates of the pool before and after the difficulty adjustment, respectively. Moreover, from Equation (3.10), we can conclude that when $t \rightarrow \infty$, the value of $R_{pool}(t)$ would converge to R_{pool} from Equation (3.6), which confirms the analysis of Eyal and Sirer of the selfish mining strategy in the long run. On the other hand, when $t < T_1$, the value of R_{pool} would be $r_{pool} \times 6t$, which confirms the analysis of Wright and Savanah of selfish mining in the short term [78].

We also have:

$$\bar{R}_{pool}(t) = \alpha \times 6t$$

Therefore:

$$\begin{aligned}
G_{pool}(t) &= \frac{R_{pool}(t) - \bar{R}_{pool}(t)}{6t} \\
&= \frac{R_{pool}(t) - \alpha \times 6t}{6t} \\
&\text{if } t > T_1 \implies \\
G_{pool}(t) &= \frac{(R_{pool} \times 6t - (R_{pool} - r_{pool}) \times 6T_1) - \alpha \times 6t}{6t} \\
&= (R_{pool} - \alpha) + (R_{pool} - r_{pool}) \times \frac{T_1}{t} \tag{3.11}
\end{aligned}$$

Figure 3.5 shows an example of $G_{pool}(t)$ for a selfish miner with hashrate of 0.35 (or %35). As can be seen in this figure, in the first period, G is negative, meaning that the pool is losing a portion of its revenue, as Wright and Savanah [78] mentioned in their paper. However, after the first difficulty adjustment, the value of G begins to increase. Somewhere around time 2,000, it reaches *zero*, meaning that at this point (which is roughly equal to 12 weeks after the beginning of selfish mining), the selfish strategy evens out. We call this point a *break-even point*. After that, it continues increasing slowly until eventually, it converges to the point that Eyal and Sirer predicted in Equation (3.6). This figure clearly shows that *time* plays an important role in the profitability of selfish mining.

3.2 Desirable Properties of a DAA

As discussed in Section 1.1.2, difficulty adjustment is a crucial mechanism to ensure the scalability of the Bitcoin network. Without difficulty adjustment, as the size of the network grows, it becomes easier to generate a new block, which imposes serious security risks on the blockchain. Because by definition, generating a Proof-of-Work

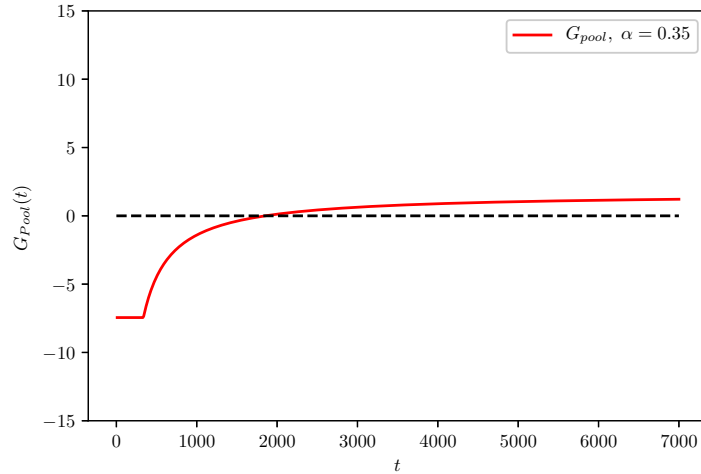


Figure 3.5: Gain for a selfish miner by time

should be hard and require a lot of effort. Otherwise, everyone could generate many consecutive blocks, which will lead to multiple branches in the blockchain and make it unstable.

On the other hand, as we discussed in Section 3.1.3, an important factor that makes selfish mining profitable is the *difficulty adjustment algorithm*. Bitcoin’s current difficulty adjustment algorithm is what makes the selfish mining attack (and some other types of mining attacks) possible because these kinds of attacks are based on wasting the computational power of other miners. In selfish mining, instead of increasing the pool’s revenue, will decrease everyone’s revenue, but decrease honest miners’ revenues more. So, from a relative perspective, it is profitable. But after the difficulty adjustment, it is going to be profitable in terms of *total net revenue*. So, when the DAA lowers the difficulty, the attackers are going to benefit from it. Hence, we need to change the difficulty adjustment algorithm, but we cannot remove it [26] [58].

We need an alternative difficulty adjustment algorithm that can maintain the scalability of the network while discouraging dishonest mining behaviour.

As mentioned before, one of our goals in this research is to find alternative difficulty

adjustment algorithms, that discourages selfish mining (and similar attacks) while allowing the network to scale.

The potential solution requires to have certain characteristics. For example Meshkov et al. describe the properties of an ideal difficulty adjustment algorithm as follows [51]:

1. “It should be resistant to known types of attacks based on difficulty manipulation.”
2. “It should lead to an almost constant desired block rate for random fluctuations in the network hashrate.”

However, the attack that Meshkov et al. had in mind was the coin-hopping attack which is different than the selfish mining attack. In the coin-hopping attack, the attacker’s focus is to alternate the difficulty between a higher value and a lower value. Then she will mine on lower difficulty periods and switches the network on higher difficulty periods. Therefore, a simple moving average DAA could achieve the goals that Meshkov et al. mentioned.

But in selfish mining, the attacker’s focus is to lower the difficulty of the whole network and gain profit from higher relative revenue. However, as discussed before, the selfish miner has to wait for a while (until the break-even point) before selfish mining becomes profitable for her. What we are trying to achieve here is to propose an alternative DAA that could extend this waiting period, to discourage miners from performing selfish mining attack. This results in a lower block generation rate and we cannot guarantee a constant block generation rate. However, there will always be a trade-off between letting miners perform selfish mining attack and tolerating a small decrease in block generation rate. Our solution aims for the latter while keeping the block generation rate as close to the desired value as possible. The reason that the block generation rate is important is that it has a direct relation

with the transaction confirmation time. Therefore, in other words, we need to keep transaction confirmation time as low as possible. Also, in addition to these two important characteristics, there are more characteristics that a difficulty adjustment algorithm should have. For example, the current difficulty adjustment algorithm is easy to understand, implement, and compute. Any proposed DAA should inherit these properties because it is highly unlikely that the Bitcoin community accepts to add a very complicated module to the Bitcoin system just for the difficulty adjustment. Also, it should not add a computationally expensive task to the system, because the mining process is a very resource-intensive task. Another important characteristic is that it should be computable by every miner in the network without a need to interact with other miners, and by just using the blockchain history and public data. Because the nature of Bitcoin is distributed and this is the core of its strength.

To summarize, based on the discussion above, a good DAA:

1. should be **resistant to selfish mining** and ideally other types of attacks based on difficulty manipulation.
2. should lead to an **almost constant block generation rate** as close to the desired value as possible.
3. should be able to **maintain the scalability** of the network.
4. should be **easy to understand, implement, and calculate**.
5. should be **independently computable by peers**.

3.3 Alternative Difficulty Adjustment Algorithms

Now that we discussed the properties of a good DAA, we propose a solution and analyze it. A solution could be a difficulty adjustment algorithm that keeps history

in the record with something like a moving average and tries to be smoother than the current algorithm.

We mentioned Bitcoin's default DAA in Chapter 1. The equation for Bitcoin's current DAA is basically as follows:

$$D_n = D_{n-1} \times \frac{\text{last period's expected time}}{\text{last period's actual time}} \quad (1.1 \text{ revisited.})$$

in which, D_n is the difficulty from the n 'th period. Based on this equation, to calculate the difficulty value of the next period, we simply calculate the ratio of expected time for the last period to the actual time of the last period, and then multiply this ratio by the difficulty value of last period. For example, we expect that a period to take 14 days. Now, assume that the last period took 13 days instead of 14 days. This means that the network difficulty is less than it should be and we have to increase it. To do so, we multiply the network difficulty by $14/13$ to get the new difficulty which is a higher value than before. This way, assuming that we do not have any change in the network's total hashrate (and of course every miner follows the protocol) the next period should take 14 days as expected.

So, we have this formula for the difficulty:

$$D_n = D_{n-1} \times \frac{2016 \times 600}{t_{b_{2016 \times n}} - t_{b_{2016 \times (n-1)}}} \quad (3.12)$$

in which, b_i is the i 'th block, and t_{b_i} is the time that block i is generated. Therefore, the denominator of the equation above shows the time difference between the last block from the last period and the last block from the period before that (2016 blocks difference). Also, the numerator is the expected time for generating 2016 blocks for an average of 600 seconds (10 minutes).

Note that all the information needed to calculate the new difficulty value is public and every peer has access to them: D_{n-1} is the current difficulty stored in the block

header¹, t_b is timestamp of a block also stored in the block header, and the numerator is a constant value of 2016×600 seconds.

3.3.1 Zeno's DAA

Let's look at the Bitcoin's DAA from another perspective. Essentially, what we calculate in Bitcoin is the expected value of difficulty for the last period. It means, for the last period, if we had used D_n as the difficulty value of period $n - 1$ instead of D_{n-1} , the period would have taken exactly 2016×600 seconds. So, in a sense, we actually calculating the expected difficulty of the last period, and hope that the next period's expected difficulty would be the same. Therefore, we call this value expected difficulty of period $n - 1$ and we denote it by E_{n-1} :

$$E_{n-1} = D_{n-1} \times \frac{2016 \times 600}{t_{b_{2016 \times n}} - t_{b_{2016 \times (n-1)}}} \quad (3.13)$$

Now, a simple modification could make the difficulties sensitive to history and not just the current period:

$$D_n = \frac{1}{2} E_{n-1} + \frac{1}{2} D_{n-1}. \quad (3.14)$$

Unlike Bitcoin's default DAA, here we do not set the difficulty of the next period to the expected difficulty of the last period. Instead, we add half of the expected difficulty and add it to half of the actual difficulty of the last period and use it for the next period. What it does is that it slows down the change in the difficulty value. Therefore, this formulation considers a history of difficulties and therefore changes more smoothly than the original one. To better show how this considers the history,

¹The actual value that is store in the block header is called *Target* and the value of difficulty can be calculated from it. [1]

we can expand Equation (3.14) as below:

$$\begin{aligned}
D_n &= \frac{1}{2} E_{n-1} + \frac{1}{2} D_{n-1} \\
&= \frac{1}{2} E_{n-1} + \frac{1}{2} \times \left(\frac{1}{2} E_{n-2} + \frac{1}{2} D_{n-1} \right) \\
&= \frac{1}{2} E_{n-1} + \frac{1}{4} E_{n-2} + \frac{1}{4} D_{n-2} \\
&= \frac{1}{2} E_{n-1} + \frac{1}{4} E_{n-2} + \frac{1}{8} E_{n-3} + \frac{1}{8} D_{n-3} \\
&= \dots \\
&\simeq \sum_i \frac{E_{n-i}}{2^i}
\end{aligned}$$

We call this new difficulty adjustment algorithm “**Zeno’s DAA**”, because it resembles *Zeno’s Paradox* in which the hare travels half of the distance between itself and the tortoise in each step, similar to our difficulty that gets adjusted to a point halfway through its final goal. Because of this property, it will take longer for the difficulty to be set to the desired amount by the selfish miners, thus it will take more time for selfish mining to become profitable. This is the type of effect we want from an alternative difficulty adjustment algorithm.

However, every alternative solution should be carefully investigated so that we can make sure it has the desired effect and also does not have any other side effect which may cause further problems. Later on, we will analyze our proposed DAA, Zeno’s DAA and present the results.

3.3.2 Zeno’s Max DAA

One of the possible downsides of Zeno’s DAA is that it gradually changes the difficulty, whether it is decreasing or increasing it. This might be good for the case that we want to extend the waiting period of the selfish mining attack, but in case

of scaling the network, it is not desirable.

So, we propose a new difficulty adjustment algorithm by making a simple change in Zeno's DAA to make it somehow asymmetric and more sensitive to increasing the difficulty than decreasing it. Here is the new DAA, called Zeno's Max DAA:

$$\begin{aligned}
 E_{n-1} &= D_{n-1} \times \frac{2016 \times 600}{t_{b_{2016 \times n}} - t_{b_{2016 \times (n-1)}}} \\
 D_n &= \max(E_{n-1}, \frac{1}{2} E_{n-1} + \frac{1}{2} D_{n-1}).
 \end{aligned}
 \tag{3.15}$$

As you can see in this formula, it chooses the difficulty value of the next period as the maximum value of E_{n-1} , and an average of E_{n-1} and D_{n-1} . In other words, we can say that Zeno's Max DAA behaves like Zeno's DAA in decreasing the difficulty and behaves like the default DAA in increasing the difficulty. Because in selfish mining, we are only concerned about the difficulty decrease, therefore in regards to selfish mining, it behaves just like Zeno's DAA. However, because it increases the difficulty all the way in one step, it scales the network as fast as the default DAA, which is much faster than Zeno's DAA.

3.3.3 Zeno's Parametric DAA

Another possible downside of Zeno's DAA is that it only calculates a simple average between E_{n-1} and D_{n-1} . We discussed that Zeno's DAA is trying to make smaller changes than the default DAA to the difficulty value to delay the profitability of selfish mining. The cost of this would be a slightly higher value for block generation time. So, there is a trade-off. In Zeno's DAA, there is no way to tune how much we can tolerate the increase in block generation time to delay the profitability of selfish mining and discourage it. By making Zeno's DAA parametric, we can overcome this

issue. Here is the formula for Zeno’s Parametric DAA:

$$\begin{aligned}
 E_{n-1} &= D_{n-1} \times \frac{2016 \times 600}{t_{b_{2016 \times n}} - t_{b_{2016 \times (n-1)}}} \\
 D_n &= (1 - \tau) \times E_{n-1} + \tau \times D_{n-1} \\
 \tau &\in [0, 1)
 \end{aligned}
 \tag{3.16}$$

The parameter here is τ . The acceptable range for this parameter is $[0, 1)$. If τ is 0.5, Zeno’s Parametric would be equivalent to Zeno’s DAA. On the other hand, if τ is 0, it would be equivalent to the default DAA, as it only includes E_{n-1} and not D_{n-1} .

The parameter τ could be selected based on the specific needs of a blockchain. For example, a particular type of blockchain might be able to tolerate longer periods of higher block generation time but wants to discourage selfish behaviour more. For this blockchain, a higher value of τ might be better, and vice versa.

3.4 Mathematical Analysis of the Solutions

The mathematical analysis, as its name suggests, uses formulas and equations to understand the effect of a proposed solution on the network. In Section 3.1.4, we analyzed the effect of Bitcoin’s default difficulty adjustment algorithm. The equation for this DAA is introduced in Equation (3.12). Similar to what we presented there, we analyze and evaluate our new proposed solutions by replacing the new formulas in the calculations and calculate the final outcomes. We introduces our proposed methods in Section 3.3 in Equations (3.14) to (3.16).

The first step to analyze any DAA is to analyze the outcome of these equations in case of selfish mining. By doing so, we will have an equation similar to Equation (3.9). The next step is to replace the result in $R_{pool}(t)$ and $G_{pool}(t)$ equations (similar to

Equations (3.10) and (3.11)). After simplifying these equations, we will have an equation for $G_{pool}(t)$ with the presence of an alternative DAA, and then we can plot the outcome (similar to Figure 3.5). Then we will have the means to compare the revenue of the selfish miner in case of the alternative DAA to its revenue in the default settings, and then we can analyze the effectiveness of the proposed DAA.

3.4.1 Bitcoin's Default DAA

We analyzed the profitability of selfish mining in section Section 3.1.4 to settle the debate. Here, we just recall it so that we can proceed to analyze our proposed methods. Suppose that the time starts at the beginning of a difficulty adjustment period when a miner with hash power α begins performing selfish mining. We also assume that no other change will happen in the network after that. Recall that when we have a selfish miner in the network, the effective hashrate of the network drop to the value of r_{total} from Equation (3.5). To shorten the formulas, we call this value ρ .

$$\rho = r_{total} = r_{pool} + r_{others} < 1 \quad (3.5 \text{ revisited})$$

Therefore, the final goal for the difficulty value would also be ρ . To recall, here is the Bitcoin's default DAA formula:

$$D_n = E_{n-1} = D_{n-1} \times \frac{2016 \times 600}{t_{b_{2016 \times n}} - t_{b_{2016 \times (n-1)}}} \quad (3.12 \text{ revisited})$$

As a result, after one difficulty adjustment, we will reach to the final difficulty value which is ρ :

$$D(t) = \begin{cases} 1 & , \quad t < T_1 \\ r_{total} = \rho & , \quad t \geq T_1 \end{cases} \quad (3.9 \text{ revisited})$$

With difficulty values calculated, now we can calculate the lengths of each period. While using Bitcoin's default DAA, only the length of the first period changes, because after that, the DAA will adjust the difficulty to its final value. Therefore:

$$\text{Period Length: } \begin{cases} T_0 = \frac{T}{\rho} \\ T_n = T, \quad \forall n \geq 1 \end{cases} \quad (3.17)$$

Just for convenience in showing the difficulty adjustment periods, we define a notation S_i to indicate the start time of period i .

$$\begin{aligned} \text{Period Start Time: } \quad S_0 &= 0 \\ S_1 &= T_0 = \frac{T}{\rho} \\ S_2 &= T_0 + T_1 = \frac{T}{\rho} + T = T(1 + 1/\rho) \\ &\dots \\ S_n &= \sum_{i=0}^{n-1} T_i = T_0 + \sum_{i=1}^{n-1} T_i = \frac{T}{\rho} + (n-1)T \\ &= T(n-1 + 1/\rho) \\ \longrightarrow T_n &= S_{n+1} - S_n \end{aligned}$$

The next step would be to calculate the revenue of the selfish pool:

$$\begin{aligned}
R_{pool}(t) &= \int_0^t \left(\frac{r_{pool}}{D(t)} \times 6 \right) dt \\
&\text{if } t > T_1 \implies \\
R_{pool}(t) &= \int_0^{T_1} \left(\frac{r_{pool}}{1} \times 6 \right) dt + \int_{T_1}^t \left(\frac{r_{pool}}{r_{total}} \times 6 \right) dt \\
&= \left[\frac{r_{pool}}{1} \times 6t \right]_0^{T_1} + \left[\frac{r_{pool}}{r_{total}} \times 6t \right]_{T_1}^t \\
&= r_{pool} \times 6T_1 + \frac{r_{pool}}{r_{total}} \times 6(t - T_1) \\
&= \frac{r_{pool}}{r_{total}} \times 6t - \left(\frac{r_{pool}}{r_{total}} - r_{pool} \right) \times 6T_1 \\
&= R_{pool} \times 6t - (R_{pool} - r_{pool}) \times 6T_1 \tag{3.10 revisited.}
\end{aligned}$$

Now that we have the value of $R_{pool}(t)$, we could calculate $G_{pool}(t)$:

$$\begin{aligned}
G_{pool}(t) &= \frac{R_{pool}(t) - \bar{R}_{pool}(t)}{6t} \\
&= \frac{R_{pool}(t) - \alpha \times 6t}{6t} \\
&\text{if } t > T_1 \implies \\
G_{pool}(t) &= \frac{(R_{pool} \times 6t - (R_{pool} - r_{pool}) \times 6T_1) - \alpha \times 6t}{6t} \\
&= (R_{pool} - \alpha) + (R_{pool} - r_{pool}) \times \frac{T_1}{t} \tag{3.11 revisited.}
\end{aligned}$$

3.4.2 Zeno's DAA

Before analyzing the network in presence of a selfish miner, let us take a look at the effect of Zeno's DAA on a network when all miners are honest. In most cases, the change in the Bitcoin network is small and gradual. Also, there is always a certain amount of randomness and uncertainty in the network that causes small changes in the network and the difficulty even without any change in the network's hashpower. Assume that the value of difficulty for the current period is 1. However, at the beginning of the period, the total hashpower of the network changes by a factor of X . For example, if 20% of the miners leave the network, the value of X would be 0.8, and if 20% more hashpower is added to the network, it would be 1.2. In this case, Bitcoin's default DAA would change the difficulty also with a factor of X . However, Zeno's DAA changes the difficulty gradually:

$$\text{Network Difficulty: } \left\{ \begin{array}{l} D_0 = 1 \\ D_1 = \frac{D_0+X}{2} = \frac{1+X}{2} = X + \frac{1-X}{2} \\ D_2 = \frac{D_1+X}{1} = \frac{\frac{1+X}{2}+X}{2} = X + \frac{1-X}{4} \\ D_3 = \frac{D_2+X}{1} = \frac{\frac{\frac{1+X}{2}+X}{2}+X}{2} = X + \frac{1-X}{8} \\ \dots \\ D_n = X + \frac{1-X}{2^n} \end{array} \right. \quad (3.18)$$

We know that even though by using Zeno's DAA, we decided to change the difficulty gradually, the goal for the value of the difficulty is also X . Given the randomness of the mining process, even by using the default DAA, we will not hit the exact value of X , we define a margin, δ , such that if the value of difficulty is in that margin of X , we consider it has hit the goal. Now let us calculate how many period it take for the difficulty of Zeno's DAA hits the goal:

$$\begin{aligned}
& X - \delta X < D_n < X + \delta X \\
\Rightarrow & |D_n - X| < \delta X \\
\Rightarrow & \left| X + \frac{1-X}{2^n} - X \right| < \delta X \\
\Rightarrow & \left| \frac{1-X}{2^n} \right| < \delta X \\
\Rightarrow & \frac{|1-X|}{\delta X} < 2^n \\
\Rightarrow & \log_2 \left(\frac{|1-X|}{\delta X} \right) < \log_2 2^n \\
\Rightarrow & \log_2 \left(\frac{1}{\delta} \times \left| \frac{1-X}{X} \right| \right) < n \tag{3.19}
\end{aligned}$$

To analyze the solution in presence of a selfish miner, similar to the previous section, we assume that the time starts at the beginning of a difficulty adjustment period when a miner with hash power α begins performing selfish mining and no other change will happen in the network after that. Also, with a selfish miner in the network, the effective hashrate of the network drop to the value of ρ (a.k.a. r_{total}). Therefore, the final goal for the difficulty value would also be ρ . However, our proposed algorithms adjust the difficulty in multiple steps instead of just one step. To recall, here is the Zeno's DAA formula:

$$E_{n-1} = D_{n-1} \times \frac{2016 \times 600}{t_{b_{2016 \times n}} - t_{b_{2016 \times (n-1)}}} \tag{3.13 revisited}$$

$$D_n = \frac{1}{2} E_{n-1} + \frac{1}{2} D_{n-1}. \tag{3.14 revisited}$$

As a result, we will have different difficulty values for each period after the first

period as follows:

$$\text{Network Difficulty: } \left\{ \begin{array}{l} D_0 = 1 \\ D_1 = \frac{D_0 + \rho}{2} = \frac{1 + \rho}{2} = \rho + \frac{1 - \rho}{2} \\ D_2 = \frac{D_1 + \rho}{1} = \frac{\frac{1 + \rho}{2} + \rho}{2} = \rho + \frac{1 - \rho}{4} \\ D_3 = \frac{D_2 + \rho}{1} = \frac{\frac{\frac{1 + \rho}{2} + \rho}{2} + \rho}{2} = \rho + \frac{1 - \rho}{8} \\ \dots \\ D_n = \rho + \frac{1 - \rho}{2^n} \end{array} \right. \quad (3.20)$$

With difficulty values calculated, now we can calculate the lengths of each period. The length of each period has a direct relation with its difficulty: the higher the value of the difficulty is, the longer the period will be. Also, it has an inverse relation with effective hashrate of the network, which is ρ . Therefore we have:

$$\begin{aligned} \text{Period Length: } T_n &= T \times \frac{D_n}{\rho} = T \times \frac{\rho + \frac{1 - \rho}{2^n}}{\rho} \\ &= T \times \left(1 + \frac{1 - \rho}{2^n \times \rho} \right) \\ &= T + T \times \frac{\frac{1}{\rho} - 1}{2^n} \end{aligned} \quad (3.21)$$

Just for convenience in showing the difficulty adjustment periods, we define a nota-

tion S_i to indicate the start time of period i .

$$\text{Period Start Time: } S_0 = 0$$

$$S_1 = T_0$$

$$S_2 = T_0 + T_1$$

...

$$S_n = \sum_{i=0}^{n-1} T_i$$

$$\longrightarrow T_n = S_{n+1} - S_n$$

Figure 3.6 shows the value of difficulty for Zeno's and the default DAA. Note that the lengths of the periods for the default DAA (as discussed in Section 3.1.4 and show in Figure 3.4) are also different from Zeno's and are shorter.

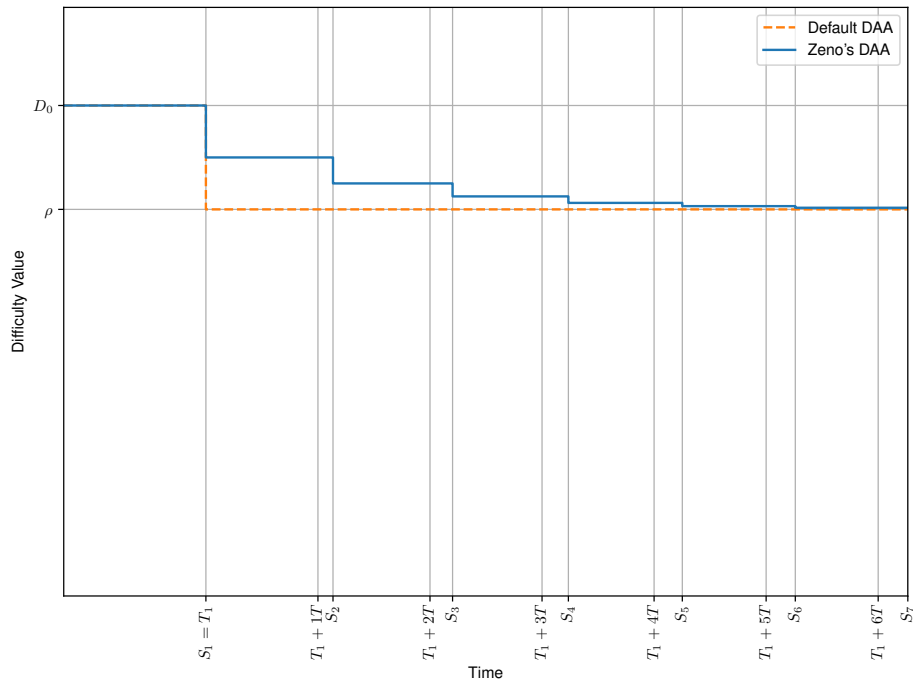


Figure 3.6: Difficulty value in time for Zeno's DAA in compare to the default DAA

The next step would be to calculate the revenue of the selfish pool, $R_{pool}(t)$, similar to Equation (3.10).

$$\begin{aligned}
R_{pool}(t) &= \int_0^t \left(\frac{r_{pool}}{D(t)} \times 6 \right) dt \\
&= \sum_{i=0}^{n-1} \int_{S_i}^{S_{i+1}} \left(\frac{r_{pool}}{D_i} \times 6 \right) dt + \int_{S_n}^t \left(\frac{r_{pool}}{D(t)} \times 6 \right) dt \\
&= 6r_{pool} \sum_{i=0}^{n-1} \int_{S_i}^{S_{i+1}} \left(\frac{1}{\rho + \frac{1-\rho}{2^i}} \right) dt + \int_{S_n}^t \left(\frac{r_{pool}}{D_n} \times 6 \right) dt \\
&= 6r_{pool} \sum_{i=0}^{n-1} \int_{S_i}^{S_{i+1}} \left(\frac{2^i}{1 + \rho(2^i - 1)} \right) dt + \int_{S_n}^t \left(\frac{r_{pool}}{D_n} \times 6 \right) dt \\
&= 6r_{pool} \sum_{i=0}^{n-1} \left[\left(\frac{2^i}{1 + \rho(2^i - 1)} \right) \times t \right]_{S_i}^{S_{i+1}} + \int_{S_n}^t \left(\frac{r_{pool}}{D_n} \times 6 \right) dt \\
&= 6r_{pool} \sum_{i=0}^{n-1} \left(\frac{2^i}{1 + \rho(2^i - 1)} \right) \times (S_{i+1} - S_i) + \int_{S_n}^t \left(\frac{r_{pool}}{D_n} \times 6 \right) dt \\
&= 6r_{pool} \sum_{i=0}^{n-1} \left(\frac{2^i}{1 + \rho(2^i - 1)} \right) \times T_i + \int_{S_n}^t \left(\frac{r_{pool}}{D_n} \times 6 \right) dt \\
&= 6r_{pool} \sum_{i=0}^{n-1} \left(\frac{2^i}{1 + \rho(2^i - 1)} \right) \times T \times \left(1 + \frac{1-\rho}{2^i \times \rho} \right) + \int_{S_n}^t \left(\frac{r_{pool}}{D_n} \times 6 \right) dt \\
&= 6Tr_{pool} \sum_{i=0}^{n-1} \frac{1}{\rho} + \int_{S_n}^t \left(\frac{r_{pool}}{D_n} \times 6 \right) dt \\
&= \frac{r_{pool}}{r_{total}} \times 6T \times n + \int_{S_n}^t \left(\frac{r_{pool}}{D_n} \times 6 \right) dt \\
&= R_{pool} \times 6T \times n + \left[\frac{r_{pool}}{D_n} \times 6 \right]_{S_n}^t \\
&= R_{pool} \times 6T \times n + (t - S_n) \times \left(\frac{r_{pool}}{D_n} \times 6 \right) \\
&= R_{pool} \times 6T \times n + (t - S_n) \times \left(\frac{r_{pool}}{\rho + \frac{1-\rho}{2^n}} \times 6 \right) \\
&= R_{pool} \times 6T \times n + (t - S_n) \times \left(\frac{r_{pool} \times 2^n}{\rho \times (2^n - 1) + 1} \times 6 \right) \\
&= R_{pool} \times 6T \times n + (t - S_n) \times \left(\frac{r_{pool} \times 2^n}{r_{total} \times (2^n - 1) + 1} \times 6 \right) \tag{3.22}
\end{aligned}$$

in which, n is the number of full DA^2 periods before t . Although it might not be straightforward to calculate n , we could try to simply it as far as we can:

$$\begin{aligned}
n &= ? \\
t &> \sum_{i=0}^{n-1} T_i \\
t &> \sum_{i=0}^{n-1} \left(T + T \times \frac{1-\rho}{2^i} \right) \\
t &> nT + \frac{1-\rho}{\rho} T \sum_{i=0}^{n-1} \left(\frac{1}{2^i} \right) \\
t &> nT + \frac{1-\rho}{\rho} T \times \left(2 - \frac{1}{2^{n-1}} \right) \\
t - \frac{1-\rho}{\rho} 2T &> nT + \frac{1-\rho}{\rho} \times \frac{T}{2^{n-1}} \\
\frac{t}{T} - \frac{1-\rho}{\rho} 2 &> n + \frac{1-\rho}{\rho} \times \frac{1}{2^{n-1}}
\end{aligned} \tag{3.23}$$

Now that we have the value of $R_{pool}(t)$, we could calculate $G_{pool}(t)$ similar to Equation (3.11).

$$\begin{aligned}
G_{pool}(t) &= \frac{R_{pool}(t) - \bar{R}_{pool}(t)}{6t} \\
&= \frac{R_{pool}(t) - \alpha \times 6t}{6t} \\
&= \frac{R_{pool} \times 6T \times n + (t - S_n) \times \left(\frac{r_{pool} \times 2^n}{r_{total} \times (2^n - 1) + 1} \times 6 \right) - \alpha \times 6t}{6t} \\
&= R_{pool} \times T \times \frac{n}{t} + \frac{t - S_n}{t} \times \frac{r_{pool} \times 2^n}{r_{total} \times (2^n - 1) + 1} - \alpha \\
&= \underbrace{R_{pool} \times T \times \frac{n}{t} - \frac{r_{pool} \times 2^n}{r_{total} \times (2^n - 1) + 1} \times \frac{S_n}{t}}_{\text{variable part}} + \underbrace{\frac{r_{pool} \times 2^n}{r_{total} \times (2^n - 1) + 1} - \alpha}_{\text{constant part}}
\end{aligned} \tag{3.24}$$

²Difficulty Adjustment

3.4.3 Zeno's Max DAA

In Section 3.3.2, we introduced Zeno's Max DAA to resolve an issue of Zeno's DAA. The issue is that Zeno's DAA treats the increment and decrements of the difficulty value the same way. However, in case we want to increase the difficulty, it is better to do it in one step instead of gradually. So, we came up with Zeno's Max DAA like this:

$$\begin{aligned} E_{n-1} &= D_{n-1} \times \frac{2016 \times 600}{t_{b_{2016 \times n}} - t_{b_{2016 \times (n-1)}}} \\ D_n &= \max(E_{n-1}, \frac{1}{2} E_{n-1} + \frac{1}{2} D_{n-1}). \end{aligned} \quad (3.15 \text{ revisited})$$

So, as discussed before, Zeno's Max DAA increases the difficulty similar to the default DAA and decreases it similar to Zeno's DAA.

On the other hand, the mathematical analysis that we performed for Zeno's DAA is for the case that we have one selfish miner in the network without any other change in the future. It means in this case, we only have the difficulty decreases. Therefore, in this scenario and with regard to the profitability of selfish mining in presence of a single selfish miner, Zeno's Max DAA behaves identically to Zeno's DAA. Therefore, everything that we calculated for Zeno's DAA in the previous section also applies to Zeno's Max DAA. So, we will not repeat the calculations here.

3.4.4 Zeno's Parametric DAA

We proposed Zeno's Parametric as an extension of Zeno's DAA to be able to create a balance between scalability and security by tuning the parameter τ . To recall from

Section 3.3.3, the equation for Zeno's Parametric DAA is:

$$\begin{aligned}
 E_{n-1} &= D_{n-1} \times \frac{2016 \times 600}{t_{b_{2016 \times n}} - t_{b_{2016 \times (n-1)}}} \\
 D_n &= (1 - \tau) \times E_{n-1} + \tau \times D_{n-1} \\
 \tau &\in [0, 1)
 \end{aligned} \tag{3.16 revisited}$$

So, we repeat the calculations that we did for Zeno's DAA, this time using Zeno's Parametric DAA equation:

$$\text{Network Difficulty: } \left\{ \begin{array}{l}
 D_0 = 1 \\
 D_1 = (1 - \tau)\rho + \tau \times D_0 = (1 - \tau)\rho + \tau \times 1 \\
 D_2 = (1 - \tau)\rho + \tau \times D_1 \\
 \quad = (1 - \tau)\rho + \tau \times ((1 - \tau)\rho + \tau) \\
 \quad = (1 - \tau)\rho + (1 - \tau)\rho \times \tau + \tau^2 \\
 D_3 = (1 - \tau)\rho + \tau \times D_2 \\
 \quad = (1 - \tau)\rho + (1 - \tau)\rho \times \tau + (1 - \tau)\rho \times \tau^2 + \tau^3 \\
 \dots \\
 D_n = \tau^n + (1 - \tau)\rho \times \sum_{i=0}^{n-1} \tau^i \\
 \quad = \tau^n + (1 - \tau)\rho \times \frac{1 - \tau^n}{1 - \tau} \\
 \quad = \tau^n + (1 - \tau^n)\rho
 \end{array} \right. \tag{3.25}$$

Now, we can also calculate the DA period length according to parameter τ :

$$\begin{aligned}
\text{Period Length: } T_n &= T \times \frac{D_n}{\rho} = T \times \frac{\tau^n + (1 - \tau^n)\rho}{\rho} \\
&= T \times \left(1 + \tau^n \left(\frac{1}{\rho} - 1\right)\right) \\
&= T + T \times \tau^n \left(\frac{1}{\rho} - 1\right)
\end{aligned} \tag{3.26}$$

Now that we have the difficulty value and length for every period, we can create a plot similar to Figure 3.6 for Zeno's Parametric. Figure 3.7 show the difficulty value for Zeno's Parametric DAA with three different values for τ . As this figure shows, in all the cases, the value of difficulty approaches the final value in multiple steps. However, different values of τ affect the heights (decrease in difficulty value) and widths (lengths of the periods) of the steps. For higher values of τ , the steps are shorter in height, meaning that the difficulty value decreases by a lower amount. Also, the steps are wider for higher values of τ , meaning that the lengths of the periods are more. Therefore, it takes more and more for the difficulty value to reach its final value, which is ρ in this case.

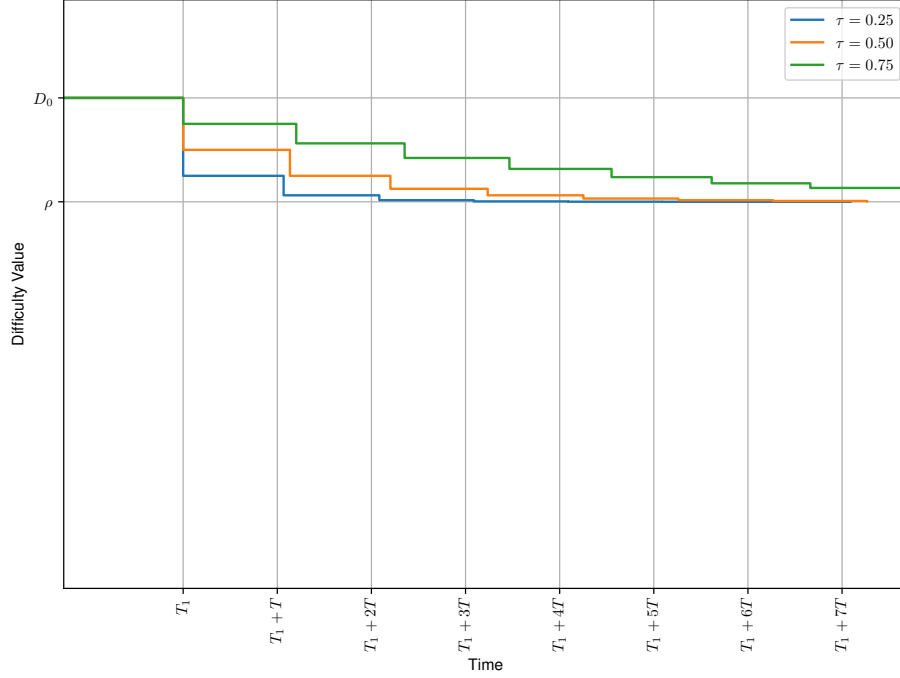


Figure 3.7: Difficulty value in time for Zeno's DAA in compare to the default DAA

The next step is to calculate the value of $R_{pool}(t)$ based in the previous calculations:

$$\begin{aligned}
R_{pool}(t) &= \int_0^t \left(\frac{r_{pool}}{D(t)} \times 6 \right) dt \\
&= \sum_{i=0}^{n-1} \int_{S_i}^{S_{i+1}} \left(\frac{r_{pool}}{D(t)} \times 6 \right) dt + \int_{S_n}^t \left(\frac{r_{pool}}{D(t)} \times 6 \right) dt \\
&= 6r_{pool} \sum_{i=0}^{n-1} \int_{S_i}^{S_{i+1}} \left(\frac{1}{\tau^n + (1 - \tau^n)\rho} \right) dt + \int_{S_n}^t \left(\frac{r_{pool}}{D_n} \times 6 \right) dt \\
&= 6r_{pool} \sum_{i=0}^{n-1} \left[\left(\frac{1}{\tau^n + (1 - \tau^n)\rho} \right) \times t \right]_{S_i}^{S_{i+1}} + \int_{S_n}^t \left(\frac{r_{pool}}{D_n} \times 6 \right) dt \\
&= 6r_{pool} \sum_{i=0}^{n-1} \left(\frac{1}{\tau^n + (1 - \tau^n)\rho} \right) \times (S_{i+1} - S_i) + \int_{S_n}^t \left(\frac{r_{pool}}{D_n} \times 6 \right) dt \\
&= 6r_{pool} \sum_{i=0}^{n-1} \left(\frac{1}{\tau^n + (1 - \tau^n)\rho} \right) \times T_i + \int_{S_n}^t \left(\frac{r_{pool}}{D_n} \times 6 \right) dt \\
&= 6r_{pool} \sum_{i=0}^{n-1} \left(\frac{1}{\tau^n + (1 - \tau^n)\rho} \right) \times T \times \frac{\tau^n + (1 - \tau^n)\rho}{\rho} + \int_{S_n}^t \left(\frac{r_{pool}}{D_n} \times 6 \right) dt \\
&= 6Tr_{pool} \sum_{i=0}^{n-1} \frac{1}{\rho} + \int_{S_n}^t \left(\frac{r_{pool}}{D_n} \times 6 \right) dt \\
&= \frac{r_{pool}}{r_{total}} \times 6T \times n + \int_{S_n}^t \left(\frac{r_{pool}}{D_n} \times 6 \right) dt \\
&= R_{pool} \times 6T \times n + \left[\frac{r_{pool}}{D_n} \times 6 \right]_{S_n}^t
\end{aligned}$$

Similar to before, the parameter n shows the number of full DA periods before t .

$$\begin{aligned}
n &= ? \\
t &> \sum_{i=0}^{n-1} T_i \\
t &> \sum_{i=0}^{n-1} (T + T \times \tau^n (\frac{1}{\rho} - 1)) \\
t &> nT + \frac{1-\rho}{\rho} \times T \times \sum_{i=0}^{n-1} \tau^n \\
t &> nT + \frac{1-\rho}{\rho} \times T \times \frac{1-\tau^n}{1-\tau}
\end{aligned}$$

The final step is to calculate $G_{pool}(t)$ based on $R_{pool}(t)$:

$$\begin{aligned}
G_{pool}(t) &= \frac{R_{pool}(t) - \bar{R}_{pool}(t)}{6t} \\
&= \frac{R_{pool}(t) - \alpha \times 6t}{6t} \\
&= \frac{R_{pool} \times 6T \times n + (t - S_n) \times (\frac{r_{pool}}{\tau^n + (1-\tau^n)r_{total}} \times 6) - \alpha \times 6t}{6t} \\
&= R_{pool} \times T \times \frac{n}{t} + \frac{t - S_n}{t} \times \frac{r_{pool}}{\tau^n + (1-\tau^n)r_{total}} - \alpha \\
&= \underbrace{R_{pool} \times T \times \frac{n}{t} - \frac{r_{pool}}{\tau^n + (1-\tau^n)r_{total}} \times \frac{S_n}{t}}_{variable\ part} + \underbrace{\frac{r_{pool}}{\tau^n + (1-\tau^n)r_{total}} - \alpha}_{constant\ part}
\end{aligned}$$

3.5 Discussion

In the previous sections, we discussed the properties of a good DAA, introduced our proposed methods, and analyzed and evaluated them mathematically. In this section, we go through various points that have to be addressed and were not covered

previously. First, we discuss the properties of Zeno’s family and whether or not they cover these properties. Also, we discuss a few other points like minimum hashrate requirement, real-world applications, etc.

3.5.1 Properties of Zeno’s Family

In Section 3.2, we introduced and discussed a set of properties that a good difficulty adjustment algorithm must possess to be a good candidate to be used in Bitcoin or other similar Proof-of-Work cryptocurrencies. In this section, we want to review those properties and discuss if our proposed DAA family has those properties or not. We covered the first two properties in form of two research questions, Q_1 and Q_2 . Here we just review the first two and then discuss the rest of the properties.

3.5.1.1 Discussion on P_1

To recall, the first property of a good DAA that we listed in Section 3.2 was that it should be resistant to selfish mining which was the whole point of our proposed methods. In the previous section, we provided a mathematical analysis of the performance of Zeno’s DAA family. Our analysis showed that substituting Zeno’s algorithms with Bitcoin’s default difficulty adjustment algorithm can successfully extend the profitability waiting period for selfish miners. This means even with enough hash power, a selfish miner should endure an extended period of time with a loss of revenue until selfish mining becomes profitable, and this is far from ideal for a specially powerful miner. Therefore we argue that Zeno’s family satisfies this property. We will also provide an evaluation by simulation for Zeno’s DAA family in Chapter 4.

3.5.1.2 Discussion on P_2

The second property of a good DAA and in fact, its main purpose is to allow the network to grow, while keeping the block generation rate fairly constant. As discussed

before, this brings up the blockchain trilemma and the trade-off between security and scalability. By using our proposed methods to increase the security (P_1), it takes a while for the average block generation time to reach its desired value. However, this will be in a tolerable range and will not have a severe effect on the network. Also, Zeno's Max improves this by responding faster to an increase in difficulty, and Zeno's Parametric could be tuned to keep this at a tolerable range. We will provide a detailed analysis of this property in Chapter 4.

3.5.1.3 Discussion on P_3

The third property, P_3 , is about maintaining the scalability of the network. The proposed DAA, similar to the default DAA, is designed to change the difficulty of the network based on the average block generation rate in the last period. Therefore, if the network scales, which will result in a lower average block generation time, Zeno's DAA will increase the network's difficulty to make up for that, just like how the default DAA does it. The only difference is that instead of doing it in one step, it does the job in multiple steps. Thus it can maintain the scalability of the network just fine.

3.5.1.4 Discussion on P_4

Our next Property is P_4 , which is about the alternative DAA being easy to understand, implement, and calculate, similar to the default Bitcoin DAA. As we discussed before, instead of setting the difficulty value to the expected difficulty value of the previous period (E_{n-1}), Zeno's DAA simply assigns it to the mid-point between the current difficulty value for the last period (D_{n-1}) and E_{n-1} . Therefore it is reasonably easy to understand. Regarding implementation and calculation of Zeno's DAA, it just adds a very simple equation (Equation (3.14)) to the computations. This equation is trivial to implement and adds an extremely insignificant and negligible

computational load to the network.

3.5.1.5 Discussion on P_5

The last property, P_5 , asks whether the alternative DAA is independently computable by peers or not? We know that Bitcoin's default DAA has this property, and peers do not need any more information from each other than what they already have. The only information that is needed to calculate the difficulty value in this case are $t_{b_{2016 \times n}}$ and $t_{b_{2016 \times (n-1)}}$ (from Equation (3.12)), which are already known to all the nodes in the network. Besides these two values, Zeno's DAA needs only one more value, D_{n-1} (from Equation (3.14)), which is the difficulty value for the previous period. Because this value is also known to all the peers, we conclude that Zeno's DAA, similar to the default DAA, is independently computable by peers and does not need any further communication between them.

3.5.2 Zeno's DAA vs. Zeno's Max DAA

Zeno's DAA is designed to change the difficulty gradually instead of in one step. But it does this regardless of the type of change (increase or decrease). However, only one of these helps us in stopping selfish miners. We want to decrease the difficulty gradually to prevent selfish behaviour. But in the case of increasing the difficulty, gradual change is not necessary. On the contrary, we can let the difficulty increase suddenly to achieve better scalability. This is the reason we introduced Zeno's Max DAA. This DAA basically behaves similar to Zeno's DAA in decrement of difficulty and similar to Bitcoin's default DAA in case of increment. Therefore, we can say that it is Zeno's DAA with better scalability. In an event of sudden growth in the network's hash power (for example for a new blockchain that is becoming popular), we can argue that Zeno's Max can better keep up with the growth than Zeno's can. However, in cases of fairly slow and gradual growth or for more stable networks (in

terms of total hash rate), the difference would not be noticeable.

3.5.3 Zeno's DAA vs. Zeno's Parametric DAA

As mentioned before, Zeno's DAA is named after Zeno's dichotomy paradox. In this paradox, the traveller (e.g. Atalanta) walks halfway to its goal in each step, pretty much what Zeno's DAA does with the difficulty value. In each adjustment, it adjusts the difficulty halfway towards the expected value (as formulated in Equation (3.14)). But why halfway? Why not third or fourth or even two-thirds of the way? This is the reason we created Zeno's Parametric, so we can choose how much to approach the goal which is the expected value of the difficulty. In a sense, the value of our parameter τ shows the remaining portion of the way towards the expected difficulty. Therefore, by setting it to *zero*, we go all the way to the goal, exactly the same as Bitcoin's default DAA. However, by changing its value, we can achieve different effects. Of course, as discussed before, it is still a trade-off between security and scalability. By increasing the value of τ , we increase the security (by extending the waiting for the attack) with the cost of decreasing the scalability of the network. Based on the characteristics and requirements of a particular blockchain, it can be decided which value of τ might be optimal to use. For a blockchain that values security very much and can handle a higher average block generation time, a higher value of τ works better and vice versa.

One pretty interesting point that we did not study in this research and might be worth considering is a variable parameter, meaning that we allow the value of τ to change during the lifetime of the blockchain. However, it is very important to define a set of rules or an algorithm, pretty much like the difficulty adjustment algorithm but for 2τ , that can be easily and independently calculated by every peer. This sure needs its own research to study the criteria for change and come up with such an algorithm.

3.5.4 Combining Zeno's Max and Zeno's Parametric

We discussed why Zeno's Max and Zeno's Parametric are good ideas and how they can improve Zeno's DAA in their own way. But what about combining them to get a better DAA? Is it possible? The answer is most certainly *yes*. These two algorithms make modifications to Zeno's DAA in two different and non-conflicting ways, and they can be easily combined to create another DAA. Let's call it Zeno's Parametric Max. This DAA could have the best of both worlds: It could be quicker in response to difficulty increase as a characteristic inherited from Zeno's Max, and also could be tuned to achieve the best balance between security and scalability based on the requirements of the blockchain as a characteristic inherited from Zeno's Parametric. For such an algorithm, all the analysis that we have done for Zeno's Max and Zeno's Parametric still applies.

3.5.5 Zeno's DAA and Selfish Mining Minimum Profitability Hashrate

Unlike some of the other proposed defences against selfish mining, our algorithm is not designed to change the minimum required hashrate for performing selfish mining. It is designed to extend the waiting time for every attacker regardless of hashrate. However, for weaker attackers like the ones closer to the minimum required hashrate (e.g. $\alpha < 0.33$) it will extend the waiting time more drastically, which means it will be a stronger force to discourage this behaviour.

3.5.6 Implementing Zeno's DAA on Existing Blockchains

Implementing a new feature or changing an algorithm in the bitcoin protocol requires forks in the blockchain. There are two types of forks: hard forks and soft forks. Soft forks are backward compatible, meaning that if some nodes do not upgrade their

software version, they can still see the blockchain as valid. Therefore, soft forks do not require the whole network to upgrade their application. In contrast, hard forks are permanent divergences from the previous versions and are not backward compatible. Therefore, they need all the nodes to upgrade to the newer version simultaneously.

Implementing Zeno's DAA is a change that requires a hard fork. Even though the implementation is straightforward, this change is not backward compatible because, in case of an increase in difficulty, the acceptable difficulty value in Zeno's DAA is lower than Bitcoin's default DAA. This means the newer blocks would not be valid for nodes with the previous version of the application that will calculate a higher difficulty value. This is not necessarily a negative point because many security patches (including other proposed defences against selfish mining) also require hard forks. But like any other hard fork, this certainly requires more effort to unite the community in favour of the change.

3.5.7 Impact of Zeno's DAA Family on Honest Miners

As discussed earlier, Zeno's DAA Family is designed to discourage miners from performing the attack and becoming selfish miners. However, what is the impact of Zeno's DAA family on honest miners? Are we punishing the whole network, including honest miners, when using Zeno's DAA family? The answer is no. In a case where we only have honest miners in the network, because of the stochastic nature of block generation in the network, there are natural fluctuations in block generation time, and it is not firmly constant. Moreover, in the normal flow of the network, the amount of change in the total hash power of the network (when miners join or leave the network) in each period is not drastic and is usually in the 1-2% range. In such cases, if we use Zeno's DAA, the difference between Zeno's DAA and default DAA is 0.5-1% which is negligible and might not even get noticed by the miners

because of said random fluctuations of block generation time. But when there is an attacker in the network, the network difficulty has to change in larger amounts e.g. 10% or even more. Here is where Zeno's DAA shows its impact. This means in a normal situation when every miner is honest, using Zeno's DAA does not have any noticeable and significant impact on the, while it still has its effect on discouraging the selfish behaviour. We will talk about this again when presenting our evaluation results.

3.6 Summary

In this chapter, we reviewed the profitability of selfish mining and showed that this attack is not profitable at first, but if the attacker is powerful enough, after a while it reaches a break-even point and starts to benefit from the attack. So, time is an important factor when it comes to analyzing the profitability of selfish mining. We showed that Bitcoin's DAA is what makes selfish mining possible. Therefore, we studied the properties of a good alternative DAA and came up with a family of solutions called Zeno's DAA family.

After introducing our proposed solutions, we provided a mathematical analysis of the solutions. To do so, similar to the profitability analysis of Bitcoin's DAA, we used the formulations for the solutions and showed their effect on the value of difficulty and extending the break-even time. Finally, in the discussion section, we discussed the properties of the solutions, how they compare, their applicability, and more. In the following section, we will use a simulation approach to evaluate our solutions and compare the results.

Chapter 4

Evaluation and Results

In the previous chapters, we talked about the problem of selfish mining and how it affects the network and harms the revenues of honest nodes in the network. Then, discussed how difficulty adjustment allows such attacks to be successful. To mitigate this problem, we proposed alternative difficulty adjustment algorithms that discourage selfish miners by extending the waiting time for selfish mining to become profitable and analysed them mathematically. In this chapter, we evaluate our proposed algorithms using a simulation approach, present the results, and discuss their effectiveness.

4.1 Research Questions

In this work, we have introduced a family of alternative difficulty adjustment algorithms that discourage selfish behaviour among miners and mining pools. In Section 3.2, we discussed the properties of a sound alternative for Bitcoin's default DAA and introduced our proposed DAA. We discussed that regardless of their computational power, selfish miners start with a loss of revenue and have to wait for a certain period to profit from selfish mining. We proposed our alternative DAAs in such a way that they extend this waiting period so that selfish miners get discouraged from

performing the attack. On the other hand, the goal of having a DAA is to keep the block generation rate constant and as close as possible to the desired block generation rate.

Below are the research questions that we address in this work. These research questions are based on one of the first two properties of a good DAA discussed previously.

(Q_1) How successful is Zeno's DAA in extending the waiting period and discouraging miners from performing selfish mining?

(Q_2) Does it keep the block generation rate constant and as close as possible to the desired rate?

4.2 Simulation Approach

In Section 3.4, we provided a mathematical analysis of the proposed solutions. Another type of analysis is to use a simulator to simulate different scenarios and analyze their outcome [59]. At the time we started to study Bitcoin and selfish mining back in 2017, there were a few Bitcoin network simulators available [28] [12]. But they were unable to serve our specific needs and/or they were very difficult to customize. So, we decided to develop our own simulator tailored to our needs.

Our simulator, which is written in Python programming language version 3, is a discrete-event simulator. Unlike continuous simulators, it is based on events. In this type of simulation, it is assumed that no changes occur between two consecutive events. So the simulator can jump to the next event and process it. These kinds of simulators are typically faster because they are not required to simulate every time slice. We used this simulator for our previous work on selfish mining, which was studying the effect of multiple selfish miners on the Bitcoin network [3]. The result of this study is published in proceedings of 2019 17th International Conference on Privacy, Security and Trust (PST).

The simulator we designed and developed here uses an event queue. This event queue is basically a priority queue, in which, the priority of every event is its occurrence time. Each step of simulation in this simulator is like this: pop the first event from the event queue, process it, and if needed, add future events that this current event may cause. Note that because we use a priority queue based on time of the events, the first event in the event queue is the first event in the future. Obviously, it is not allowed to add an event from the past. So for every event that you add to the queue, it should have a time greater than the current event time.

In our Bitcoin simulator, we have two kinds of events: 1) the event of finding/generating a new block by a miner (mining event), 2) the event of receiving a new block by a miner that is found/generated by another miner (receiving event). Bitcoin protocol clearly states the action that should be taken in the case of each event.

For the first one, when a node finds a new block, it should broadcast it to the rest of the nodes. Therefore, upon processing this event, we will add a *receive event* for all other nodes in the network, considering a propagation delay that comes from a gamma distribution. Details of this part of implementation are derived from the work of Decker and Wattenhofer [17] in 2013, in which they analyzed information propagation in the Bitcoin network. Also, we will schedule another mine event for the current node, x seconds into the future in which, x comes from an exponential distribution. Note that the scale parameter of this exponential distribution has an inverse relationship with the node's hashrate, α .

The second type of event is receiving a newly generated block from another miner. In this case, the same as what protocol says, the miner will accept the new block (of course, if it is valid) and starts to mine on top of that. It means we will schedule a new *mine event* for the node in the future.

For implementing other mining strategies, we just have to define different types of miners and implement the policy of the miner on each event under different cir-

cumstances. For the selfish mining attack, we implemented the strategy described in Section 2.3 and used this new miner for our analysis. We also extended this simulator to simulate our proposed difficulty adjustment algorithms and study their effect.

An overview of the workflow of the simulator is shown in Figure 4.1. For each scenario that we need to simulate, a simulation configuration has to be created. This configuration consists of the number of miners, their hash power, difficulty adjustment algorithm, episode length, and any other variables that define this scenario. Based on this configuration, we create multiple instances of the simulator class. Then we assign these simulator instances to different processors (using Python’s multiprocessing capabilities) to run them in parallel and increase efficiency. After running the simulator instances for the desired amount of time, we will get one simulation result object per simulation instance. The next step is to aggregate them (e.g. calculating the average, etc.). Then the simulator will provide a report from the aggregated results in the form of figures or text. This will lead to the final step which is interpreting the results and analyzing the effect of our proposed algorithms on the profitability of selfish mining. This can be further proof of our mathematical analysis. For more technical details about our simulator, check Appendix A.

4.3 Simulation Results

In this section, we use our simulator to evaluate the results of our proposed methods. We start with the effect of selfish mining on the network with the default DAA. Then we evaluate our proposed methods and comparing them to the default DAA in terms of profitability of selfish mining.

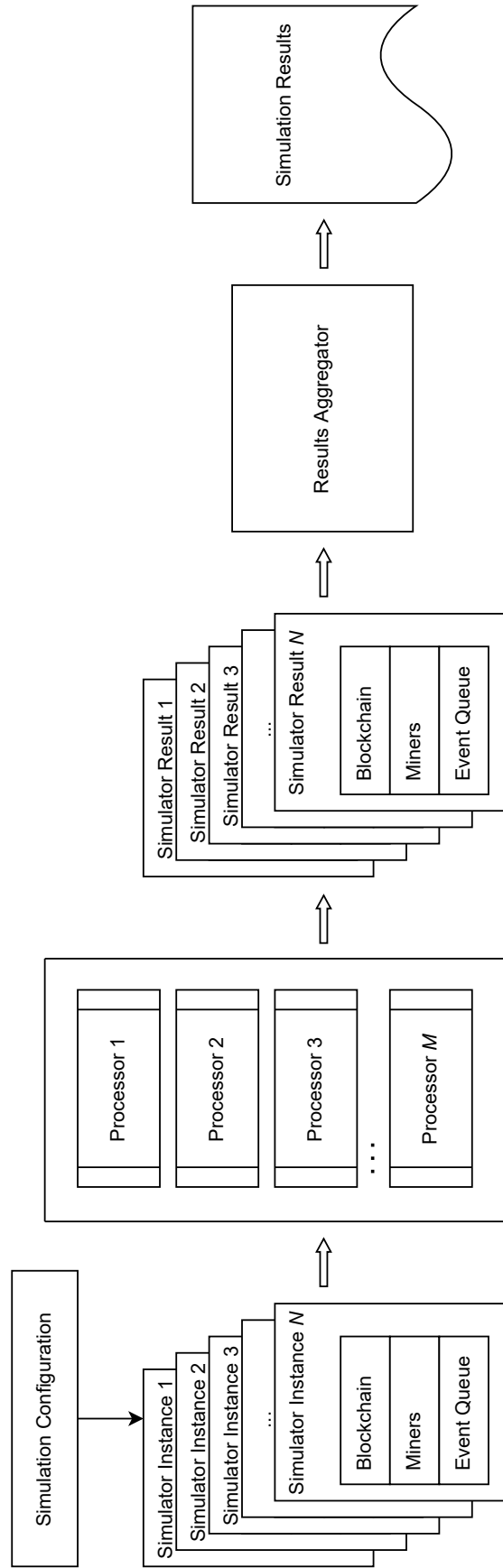


Figure 4.1: A simplified overview of the simulator's workflow.

4.3.1 Selfish Mining

In this section, we use our simulator to analyze the Bitcoin's network, first to observe the effect of selfish mining on the network, and second to study the effect of our proposed DAA to see how it performs and whether it is a suitable alternative for the default DAA or not. So, besides the default DAA, we also implemented our proposed DAA, and we will compare the results later on.

For the case that all the miners are honest and loyal to the protocol, Figure 4.2 show the outputs of Equation (3.2) and also the results of the simulation¹, which demonstrates $R_{P_i}(t)$ from Equation (3.3), for different values of α . As shown in these figures, the two curves in both figures match perfectly.

$$\bar{R}_{P_i}(t) \approx \alpha_i \times t \times 6 \quad (3.2)$$

$$R_{P_i}(t) \approx \bar{R}_{P_i}(t), \text{ if all miners are honest} \quad (3.3)$$

The desired value for $G_{P_i}(t)$ from Equation (3.4) when all miners are honest is *zero*. Figure 4.3 shows this value for honest miners with different hashrates.

$$G_{P_i}(t) = \frac{R_{P_i}(t) - \bar{R}_{P_i}(t)}{t \times 6} \quad (3.4)$$

The next result of the simulator is related to the Equation (3.7) from Section 3.1.4:

$$T = 2016 \times \frac{1}{6} = 336 \text{ hours} \quad (3.7 \text{ revisited})$$

In an *all-honest scenario*, we expect the value of T to be very close to 336 hours. We

¹All the simulation results here are average between 35 independent simulations.

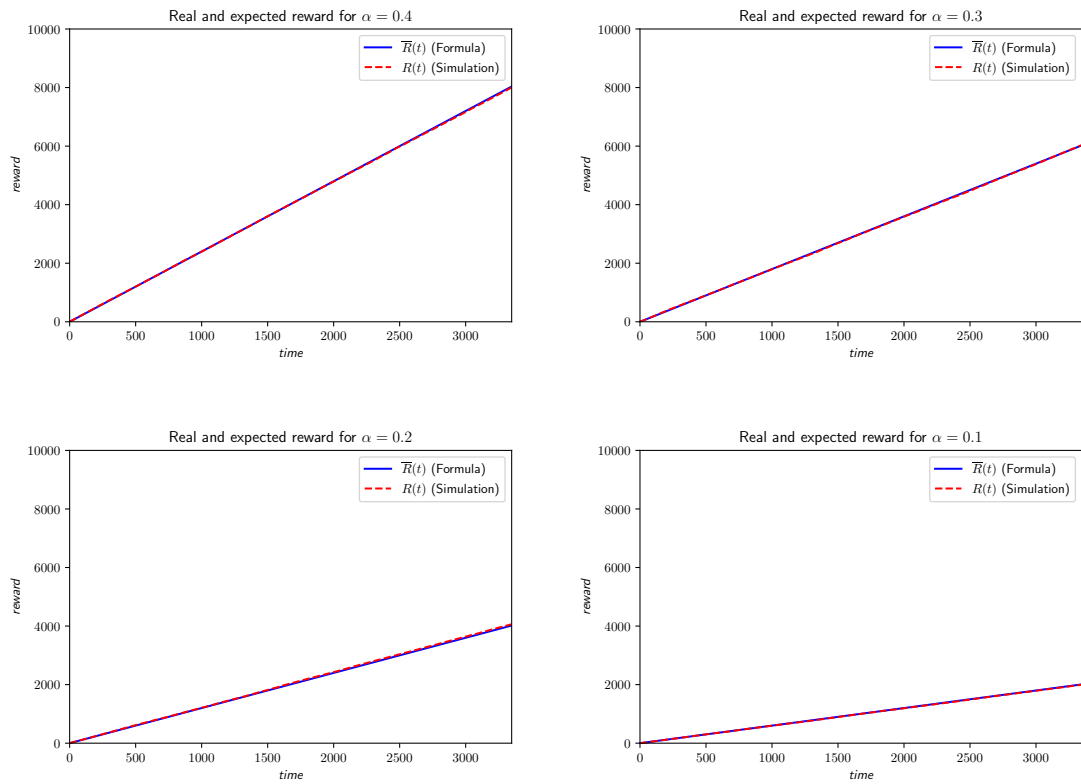


Figure 4.2: Real and expected rewards for honest miners with different hashrates

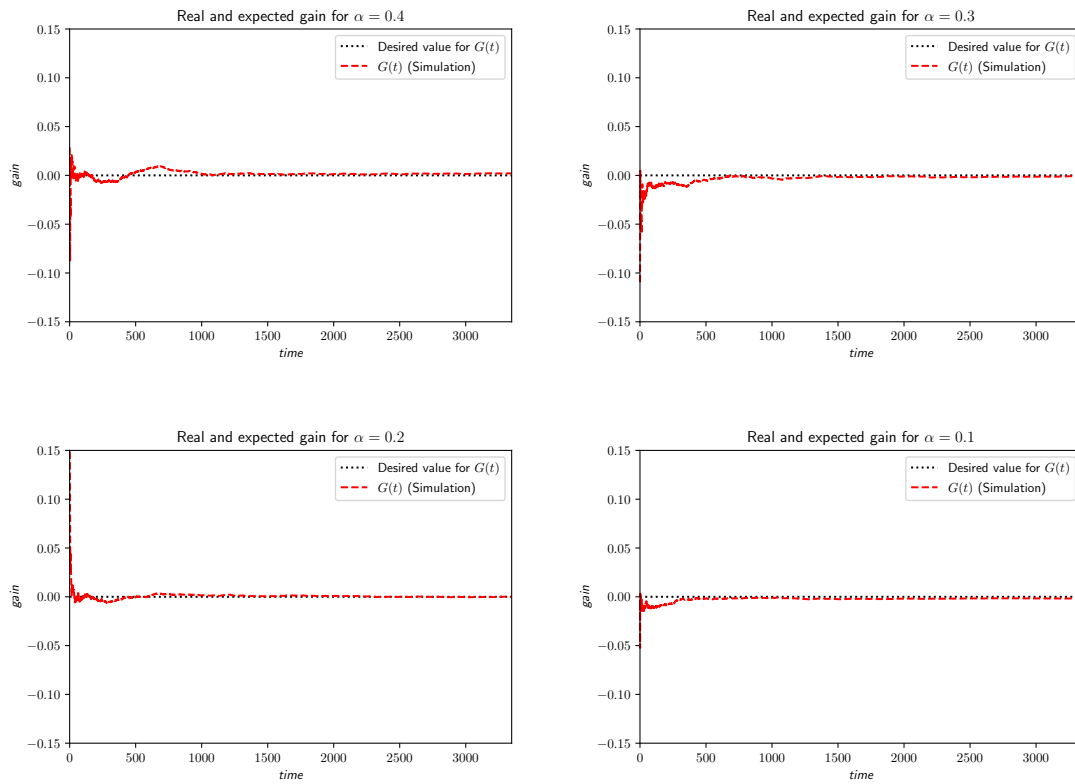


Figure 4.3: Real and expected gain for honest miners with different hashrates

ran the simulator 350 times and Figure 4.4 shows the histogram of different values of T from these simulations. Also, the mean and variance of the samples are as follows:

$$\mu = E[T] = 336.22 \quad (4.1)$$

$$\sigma^2 = Var(T) = 63.76 \quad (4.2)$$

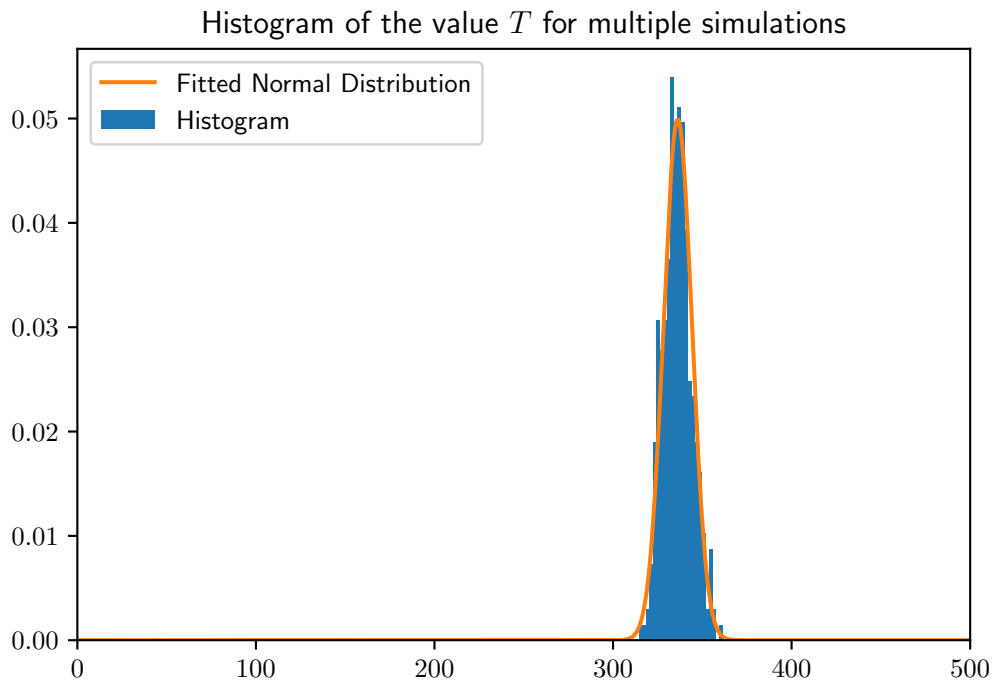


Figure 4.4: Histogram of value T

The next equation in Section 3.1.4 is T_1 , which is the length of the first difficulty adjustment period in the presence of a selfish miner. Figure 4.5 shows the value of T_1 for different network configurations with selfish miners who have hashrates between 0.25 and 0.50. As you can see, the two curves match perfectly.

$$T_1 = \frac{T}{r_{total}} > T \quad (3.8 \text{ revisited})$$

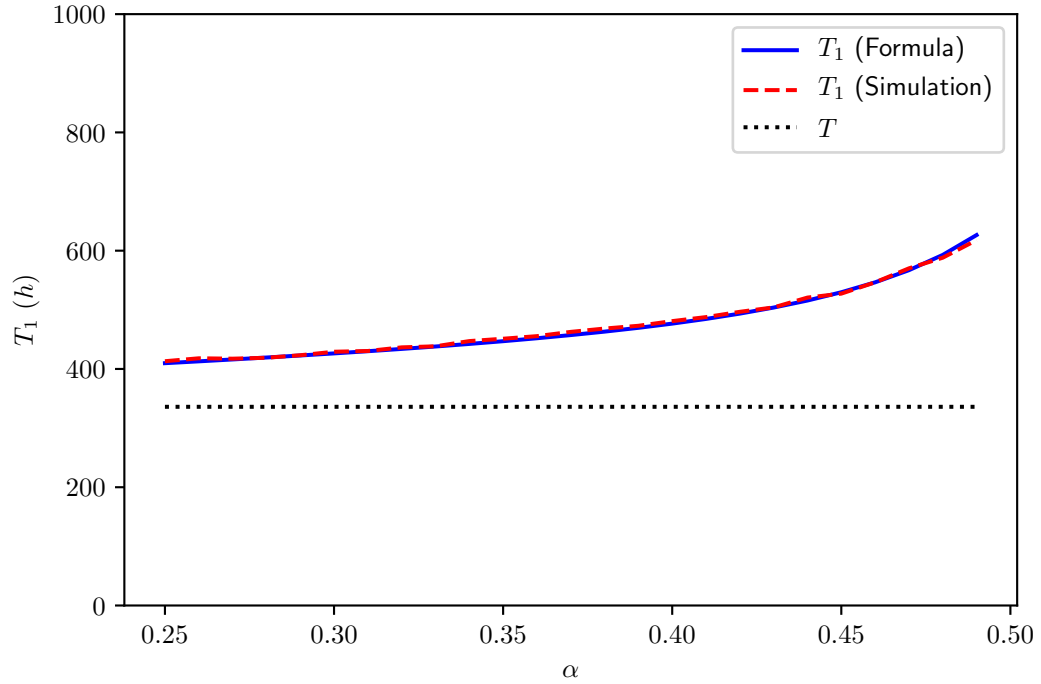


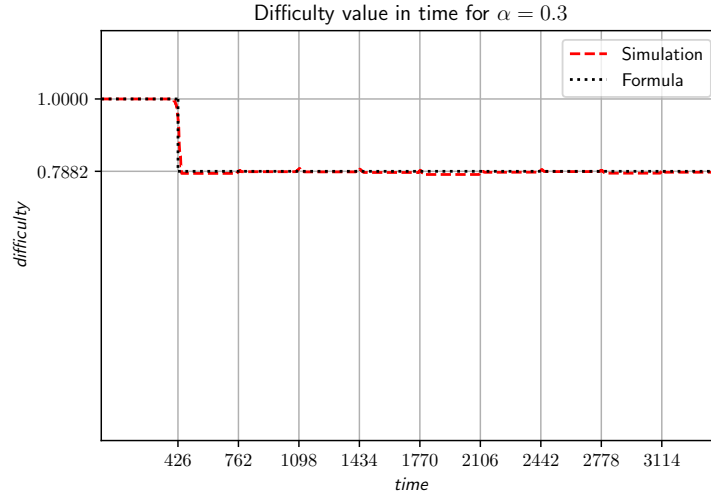
Figure 4.5: Value of T_1 for different values of α

In Equation (3.9) we have the value of *the difficulty*, in the case that the selfish miner starts at time *zero* with no other change in the network.

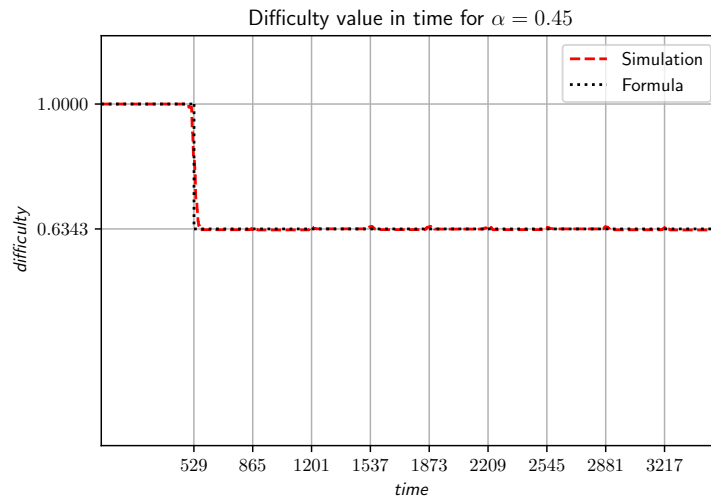
$$D(t) = \begin{cases} 1 & , t < T_1 \\ r_{total} & , t \geq T_1 \end{cases} \quad (3.9)$$

Figures 4.6a and 4.6b show $D(t)$ for two different network configurations with $\alpha = 0.30$ and $\alpha = 0.45$ respectively. Again in this figure, the two curves match perfectly. It is important to note that because of the different values of α , both T_1 and the

final value of D are different in the two figures (T_1 has the direct relationship with α and the final value of D has an inverse relationship with it).



(a) $\alpha = 0.30$



(b) $\alpha = 0.45$

Figure 4.6: D for different values of α

The next equation is Equation (3.10) which is defined as follows:

$$R_{pool}(t) = R_{pool} \times 6t - (R_{pool} - r_{pool}) \times 6T_1 \quad (3.10 \text{ revisited})$$

Figures 4.7 and 4.8 show the values of $R(t)$ (from simulation and formula) and $\bar{R}(t)$ (expected value of reward in all honest scenario) for two network configurations of $\alpha = 0.40$ and $\alpha = 0.33$ respectively.

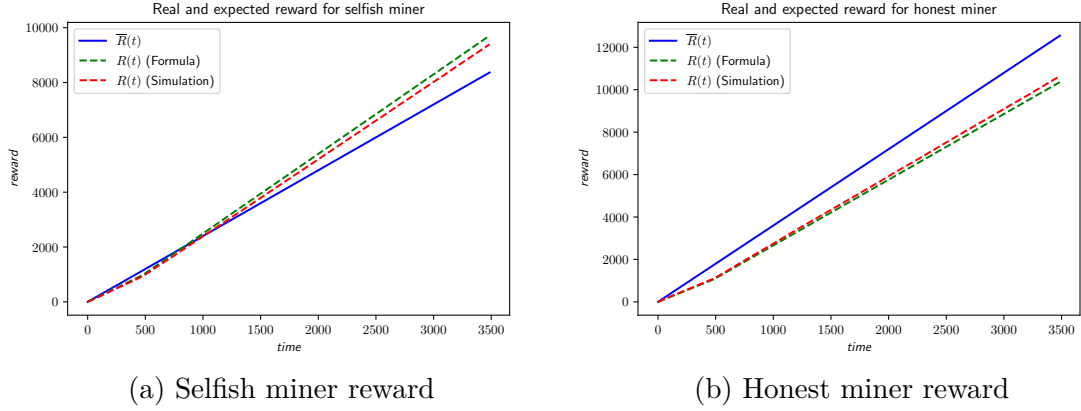


Figure 4.7: Rewards for miners when $\alpha = 0.40$

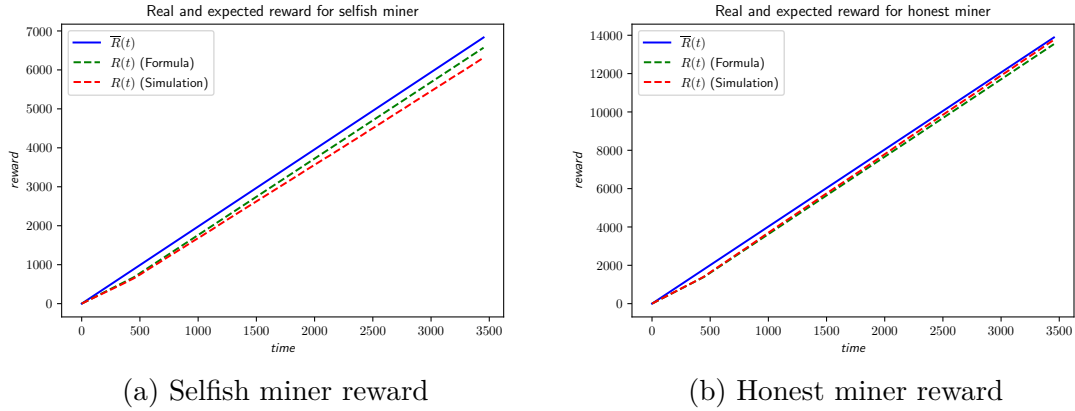


Figure 4.8: Rewards for miners when $\alpha = 0.33$

The last equation, which is the most important for us, is Equation (3.11):

$$G_{pool}(t) = (R_{pool} - \alpha) + (R_{pool} - r_{pool}) \times \frac{T_1}{t} \quad (3.11)$$

Similar to the previous figures, Figures 4.9 and 4.10 show the values of $G(t)$ from simulation and formula for two network configurations of $\alpha = 0.40$ and $\alpha = 0.33$ respectively. The black dotted line shows the line $G(t) = 0$ which is the expected

value of gain in all-honest scenario.

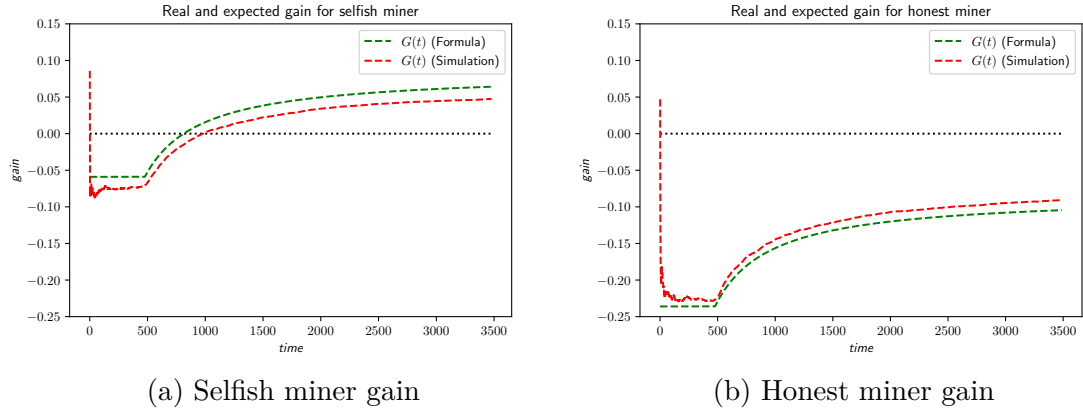


Figure 4.9: Gain for miners when $\alpha = 0.40$

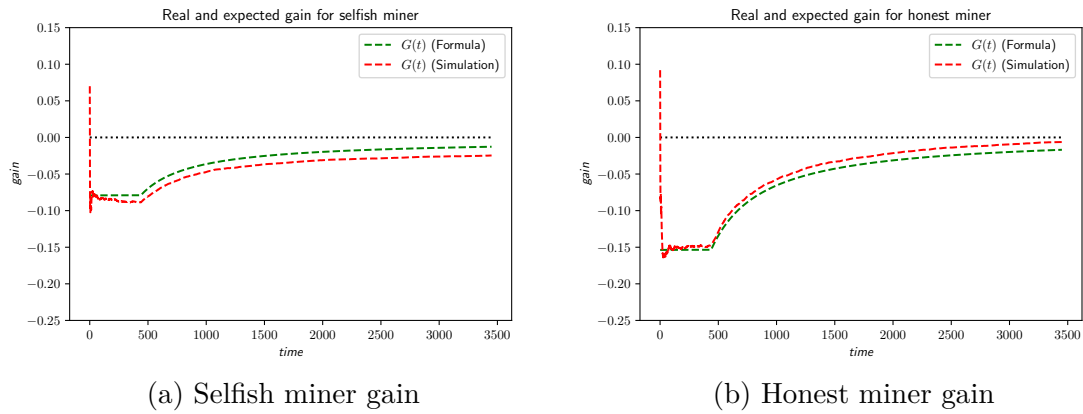


Figure 4.10: Gain for miners when $\alpha = 0.33$

4.3.2 Profitability of Selfish Mining with Multiple Attackers

In their paper [21], Eyal and Sirer supposed that there is a pool of selfish miners who work together, and on the other hand, there are the rest of the nodes who mine honestly and follow the Bitcoin protocol faithfully. They formulated their problem

as follows:

Total mining power of the whole network : 1

Total mining power of the selfish miner : α

Total mining power of all honest miners combined : $1 - \alpha$

The ratio of honest miners accepting selfish branch : γ

But what if there are two (or even more) competitive selfish pools in the network that are not willing to cooperate? How can this affect the revenue of those pools and other honest miners in the network? Can this conflict neutralize their effort to increase their revenue? What if one of those pools is bigger than the other? Can the bigger pool defeat the smaller pool? Does knowing about each other affect their strategy of mining? These are some interesting question about having multiple competitive selfish pools and we think it is worth working on them and finding the answers. So we are formulating the problem as follows:

T : Total number of pools

N : Number of selfish pools

$\mathbb{P} = \{P_i \mid i = 0, \dots, T - 1\} = \mathbb{H} \cup \mathbb{S}$: Set of all miners

$\mathbb{H} = \{H_i = P_i \mid P_i \text{ is honest}\}$: Set of honest miners

$\mathbb{S} = \{S_i = P_i \mid P_i \text{ is selfish}\}$: Set of selfish miners

α_i for $i = 0, \dots, T - 1$: Hashrate of P_i

Also, in general, we denote every miner as P (for peer), instead of using S or H . So a miner P_i can either refer to a selfish miner or honest miner.

In the particular case of having just two selfish pools and one honest pool in play, based on the definition above, we define two parameters. The first parameter is α_t

which is the total mining power of two selfish pools combined, and another parameter δ that reflects the difference between their mining power. The formal definition is as follows:

$$\begin{aligned}\mathbb{P} &= \{H_0, S_1, S_2\} \\ N &= 2 \quad (\text{Two selfish miners, } S_1 \text{ \& } S_2) \\ \alpha_t &= \alpha_1 + \alpha_2 \quad \text{where} \\ \alpha_1 &= \frac{\alpha_t}{2} \times (1 + \delta), \quad \alpha_2 = \frac{\alpha_t}{2} \times (1 - \delta)\end{aligned}$$

and of course for the honest pool we have:

$$\begin{aligned}|\mathbb{H}| &= 1 \quad (\text{One honest miner, } H_0) \\ \alpha_0 &= 1 - \alpha_t = 1 - (\alpha_1 + \alpha_2)\end{aligned}$$

So in this case, we want to study the effect of different values for α_t (the total mining power of two selfish pools) and δ (the difference in their mining power) in an environment with two selfish pools.

First, we start with two selfish miners, and then expand it to more selfish miners.

4.3.2.1 Two competitive selfish miners

For the first scenario, we are going to take a look at a very simple network hash power configuration. For the sake of simplicity, from now on, we suppose that the number of honest pools is one. It means that we treat all individual honest miners or honest pools as one big pool of honest miners.

Let's take a look at a simple example with one honest miner and two selfish miners:

$$\mathbb{P} = \{H_0, S_1, S_2\}$$

$$H_0 : \alpha_0 = 0.49, S_1 : \alpha_1 = 0.17, S_2 : \alpha_2 = 0.34$$

$$\Rightarrow \alpha_t = 0.51 \quad \text{and} \quad \delta = 1/3$$

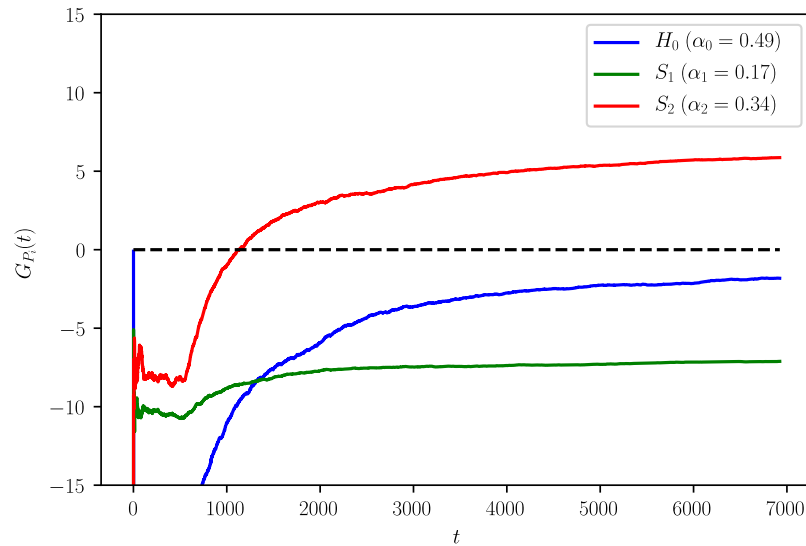


Figure 4.11: Two selfish miners' gains, when one is twice stronger than the other.

This is an interesting example. As can be seen in the figure Figure 4.11, the gain is above zero (around +5%) for the more powerful selfish miner, S_2 , but for other miners, S_1 , and H_0 it is harmful and they lose revenue, and the weaker selfish miner even loses more.

It gets more interesting when one compares the result with the same configuration, except the weaker selfish miner decides to stay honest instead. Figure 4.12 shows the results of this configuration. From the perspective of the second selfish miner (which is an honest miner now), it is clear that in presence of a more powerful selfish miner, the selfish strategy not only decreases his revenue but also helps the stronger selfish miner to increase her revenue. Another point worth mentioning is that even in the first scenario with two selfish miners, based on Figure 4.11 it takes approximately

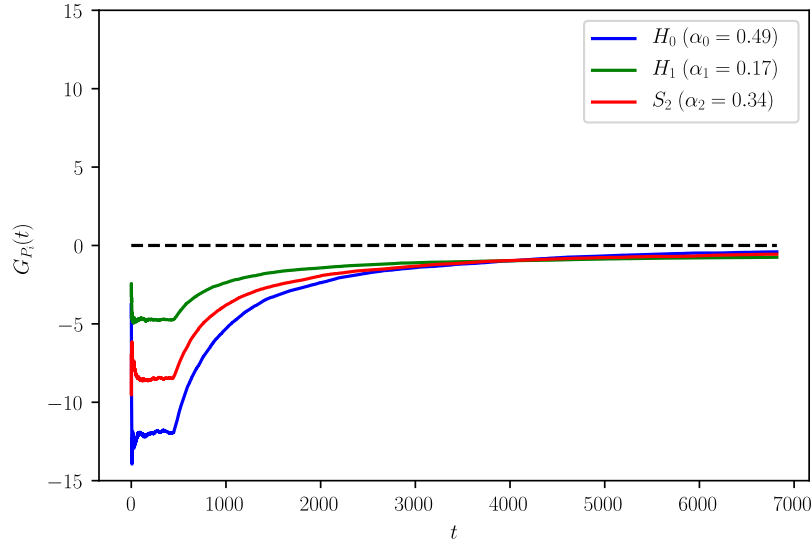


Figure 4.12: The gains when the weaker selfish miner in Figure 4.11 turns honest.

1150 hours (or more than 48 days) for the stronger selfish miner to recover her lost revenue. So again, it is not profitable in the short term.

For further analysis of the case with two selfish miners (with the rest of the network as an honest miner), we used the same scenario and changed the values of α_t and δ in ranges of $[0.05, 0.95]$ and $[0.0, 0.4]$ respectively. In this scenario, we have S_1 and S_2 , and their hash powers are $\alpha_1 = \frac{\alpha_t}{2} \times (1 - \delta)$ and $\alpha_2 = \frac{\alpha_t}{2} \times (1 + \delta)$, respectively. Provided that S_2 is always the more powerful than S_1 , we call it *stronger selfish miner*. Likewise, we call S_1 *weaker selfish miner*. Figure 4.13 depicts the result of this simulation. Sub-figure (a) shows G for the stronger selfish miner, and sub-figure (b) shows it for the weaker one. Also, positive values of G has been shown in green and negative values has been shown by red.

As it is shown in these figures, for the weaker selfish miner, the green area is very small. It means that there are very limited situations that can result in the profitability of selfish mining in case there is a stronger selfish miner in the network. To be more specific, only when the total power of two selfish miners is around 55% and more ($\alpha_t > 55\%$), and the difference factor is very small ($\delta < 0.04$), selfish mining is profitable for the weaker selfish miner. In every other condition, selfish mining is

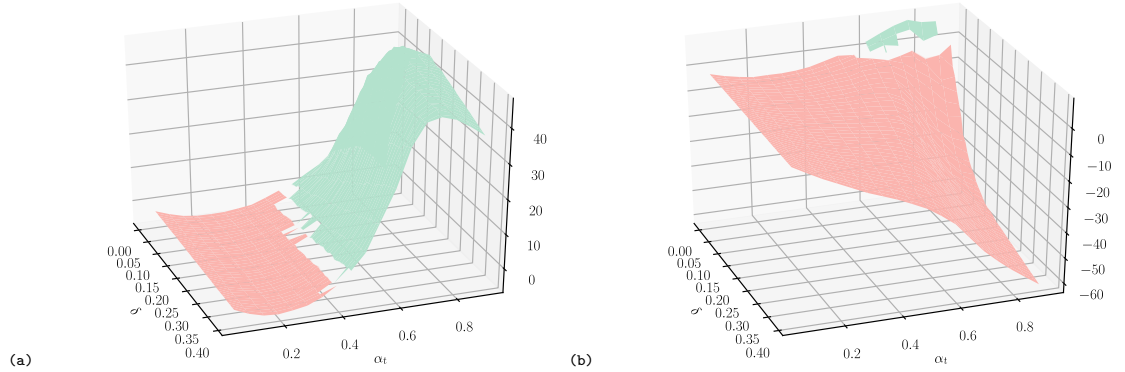


Figure 4.13: Results of simulation for two selfish miners and one honest miner with the values of α_t and δ in ranges of $[0.05, 0.95]$ and $[0.0, 0.4]$ respectively. Sub-figure (a) shows the results for the stronger one (S_2) and sub-figure (b) shows them for the weaker one (S_1). Green and red colors show positive and negative values for the G , respectively.

harmful to her revenue. One other interesting point is that in case $\delta > 0.5$, when the two selfish miners become more powerful ($\alpha_t > 0.5$), the loss for the weaker one increases.

On the other hand, for the stronger selfish miner, the profitability limit for α_t is around 0.5. It means that if a miner wants to start mining selfishly, first, there should be another weaker selfish miner in the play, and second, the total hash power of them together should be around half of the hash power of the network. Also, the line which divides green color and red color in Figure 4.13(a) is a little oblique. It means for the low values of δ , $\alpha_t \approx 0.55$, and for the higher values of δ , $\alpha_t \approx 0.4$. This makes sense because when we have higher values for δ , it means that S_1 is actually more powerful.

4.3.2.2 Three selfish pools

For the case with three selfish miners, we analyze the problem from the perspective of one of them, namely S_1 . Assume that there are three selfish miners, S_1 , S_2 , and S_3 , and also an honest miner, H_0 . Let's assume that α_0 and α_1 (hash powers of H_0 and S_1) are constant. Having two competitors, the best case scenario for S_1 is when

the two competitors share the rest of hash power evenly. On the other hand, the worst case scenario for S_1 is when one of the competitors is dominant, meaning she possesses the rest of the network hash power, while the other possesses nothing. This is equivalent to the case that there is just two selfish miners, S_1 and S_2 . Because there are limitations in visualization of three selfish miner's case, we focus on the worst case and best case scenarios for the S_1 .

$$\mathbb{P} = \{H_0, S_1, S_2, S_3\}$$

$$H_0 : 0.3 \leq \alpha_0 \leq 0.6, \quad S_1 : 0.2 \leq \alpha_1 \leq 0.3$$

$$\text{Worst case: } \alpha_3 = 0, \quad \alpha_2 = 1 - \alpha_0 - \alpha_1$$

$$\text{Best case:} \quad \alpha_2 = \alpha_3$$

Tables 4.1 to 4.4 show the results of this configuration. In these tables, the top row is the α_0 and the left column in the α_1 , and every number in the table is the amount of G_{S_1} which is the gain for selfish miner S_1 . Negative numbers (loss) are shown in shades of red and positive numbers (gains) are shown in shades of green. Tables 4.1 and 4.2 show the results for the worst cases scenario. The first one shows the G_{S_1} in the short term (approximately one month) and the second one shows it in the long term (approximately ten months). Similar to the previous scenarios with one or two selfish miners, in short term, S_1 will not profit from selfish mining. However, in long term, when S_1 has nearly 30% and the H_0 has between 40% and 55%, selfish mining becomes slightly profitable for S_1 . In this situation, H_0 is neither strong enough to overpower selfish miners, nor weak enough to let the S_1 's competitor, S_2 , become too strong.

In Tables 4.3 and 4.4, the best case scenario has been shown. As these tables show when S_1 becomes stronger and also H_0 becomes weaker (which implies that S_2 and S_3 are growing stronger but not stronger than S_1) the selfish strategy is profitable

Table 4.1: Worst case of S_1 's gain in the short term. The numbers in the first column show α_1 and the numbers in the first row show $\alpha_2 + \alpha_3$. Numbers in the table shows the change in S_1 's revenue in this scenario. In this case, all of the cells show a loss.

	0.30	0.35	0.40	0.45	0.50	0.55	0.60
0.20	-19.97	-19.76	-12.86	-8.82	-7.43	-6.63	-6.45
0.21	-20.96	-20.62	-12.40	-8.85	-7.12	-6.45	-6.23
0.22	-21.96	-20.80	-11.65	-8.20	-6.78	-6.24	-6.13
0.23	-22.96	-19.87	-11.15	-7.82	-6.03	-5.70	-5.67
0.24	-23.92	-19.50	-10.18	-7.15	-5.77	-5.33	-5.23
0.25	-24.89	-18.61	-10.00	-6.63	-5.27	-4.80	-5.11
0.26	-25.89	-17.62	-8.67	-5.89	-4.96	-4.70	-4.63
0.27	-26.86	-16.74	-7.48	-4.75	-3.91	-4.17	-4.58
0.28	-27.81	-15.77	-6.15	-4.38	-3.44	-3.59	-4.33
0.29	-28.65	-13.21	-4.86	-3.27	-2.85	-3.27	-3.83
0.30	-29.65	-11.85	-3.98	-2.38	-2.00	-2.67	-3.48

Table 4.2: Worst case of S_1 's gain in the long term. The numbers in the first column show α_1 and the numbers in the first row show $\alpha_2 + \alpha_3$. Numbers in the table shows the change in S_1 's revenue in this scenario. In this case, most of the cells show a loss except a few with $\alpha_1 \geq 0.29$.

	0.30	0.35	0.40	0.45	0.50	0.55	0.60
0.20	-19.99	-19.95	-11.29	-7.29	-5.81	-5.25	-5.21
0.21	-20.99	-20.92	-10.56	-6.75	-5.40	-4.77	-4.94
0.22	-21.99	-21.20	-9.74	-6.09	-4.68	-4.34	-4.61
0.23	-22.99	-19.55	-8.82	-5.32	-4.00	-3.92	-4.22
0.24	-23.98	-18.38	-7.79	-4.53	-3.41	-3.42	-3.67
0.25	-24.98	-16.71	-6.66	-3.67	-2.88	-2.83	-3.55
0.26	-25.98	-15.14	-5.51	-2.73	-1.95	-2.21	-2.86
0.27	-26.97	-13.68	-3.99	-1.47	-1.15	-1.60	-2.45
0.28	-27.96	-11.49	-2.29	-0.50	-0.37	-0.91	-1.86
0.29	-28.93	-9.56	-0.74	0.70	0.54	-0.28	-1.35
0.30	-29.93	-7.19	0.96	1.87	1.44	0.33	-1.00

for S_1 . This is similar to the case with two selfish miners when the weaker selfish miner increases the stronger one's revenue while losing some part of her revenue. It seems that in cases that selfish miners, regardless of how many they are, have the majority of the mining power, the strongest selfish miner will profit, in the expense

Table 4.3: Best case of S_1 's gain in the short term. The numbers in the first column show α_1 and the numbers in the first row show $\alpha_2 + \alpha_3$. Numbers in the table shows the change in S_1 's revenue in this scenario. In this scenario, in cases of $\alpha_1 \geq 0.29$ and $\alpha_2 + \alpha_3 \leq 0.45$, S_1 could gain a profit.

	0.30	0.35	0.40	0.45	0.50	0.55	0.60
0.20	-7.69	-5.93	-5.45	-5.21	-4.94	-5.28	-5.45
0.21	-7.04	-5.64	-4.94	-4.38	-4.73	-4.79	-5.44
0.22	-6.23	-4.91	-4.30	-4.10	-4.17	-4.73	-5.33
0.23	-5.53	-4.20	-3.72	-3.25	-3.75	-4.17	-4.88
0.24	-5.03	-3.66	-3.16	-3.26	-3.40	-4.03	-4.66
0.25	-3.56	-2.71	-2.37	-2.96	-3.16	-3.71	-4.25
0.26	-2.77	-1.87	-1.89	-1.67	-2.44	-3.46	-3.93
0.27	-1.77	-0.95	-0.91	-1.34	-2.08	-2.73	-3.97
0.28	-0.07	-0.22	-0.41	-0.59	-1.16	-2.24	-3.79
0.29	0.83	1.00	0.99	0.16	-0.55	-2.06	-3.26
0.30	2.43	2.11	1.21	0.56	-0.52	-1.75	-3.09

of the weaker ones lose some of theirs. Also, by comparing Tables 4.3 and 4.4, it can be seen that in the long term, the number of configurations that are profitable for S_1 increases, because similar to previous examples, it takes a long time for the S_1 's revenue to converge to its final value.

Figure 4.14 also shows the results of the Table 4.3 in the form of a graph. Every curve corresponds to a row in the table. This figure depicts how different values of α_0 and α_1 affect the profitability of selfish mining for S_1 . Note that this figure is based on the best case scenario in which $\alpha_2 = \alpha_3$.

4.3.2.3 Final Remarks on Selfish Mining with Multiple Attackers

In this section, we addressed a problem regarding competitive selfish mining in Bitcoin. First, we start to explain the main problem of this work and define it formally, similar to what Eyal and Sirer did in their own work introducing selfish mining. The main question is rather simple: “What if there are two or even more selfish pools in the network, who are not willing to join/cooperate?”

Table 4.4: Best case of S_1 's gain in the long term. The numbers in the first column show α_1 and the numbers in the first row show $\alpha_2 + \alpha_3$. Numbers in the table shows the change in S_1 's revenue in this scenario. In this scenario, in cases S_1 being more powerful and S_2 and S_3 , who have equal hashrates are not as powerful as S_1 , it will be profitable for S_1 .

	0.30	0.35	0.40	0.45	0.50	0.55	0.60
0.20	-4.20	-2.99	-2.60	-2.62	-2.86	-3.44	-4.12
0.21	-3.31	-2.13	-1.83	-1.88	-2.42	-3.03	-3.84
0.22	-2.30	-1.23	-0.96	-1.41	-1.76	-2.55	-3.57
0.23	-0.99	-0.17	-0.13	-0.48	-1.20	-2.05	-3.14
0.24	0.36	0.85	0.79	0.14	-0.72	-1.70	-2.81
0.25	2.00	2.10	1.77	0.92	0.00	-1.25	-2.33
0.26	3.59	3.33	2.69	1.99	0.65	-0.62	-2.03
0.27	5.44	4.70	3.86	2.68	1.29	0.08	-1.64
0.28	7.22	6.06	4.94	3.74	2.35	0.60	-1.21
0.29	9.02	7.77	6.33	4.67	3.07	1.15	-0.76
0.30	11.30	9.33	7.46	5.72	3.88	1.76	-0.37

First, we start with two selfish miners in the network. We start from a simple scenario, then expand it to more complex scenarios, and discuss the results. For the case of two selfish miners, we present simulation results for a wide range of parameters, α_t and δ . With analyzing these results, we find some interesting points.

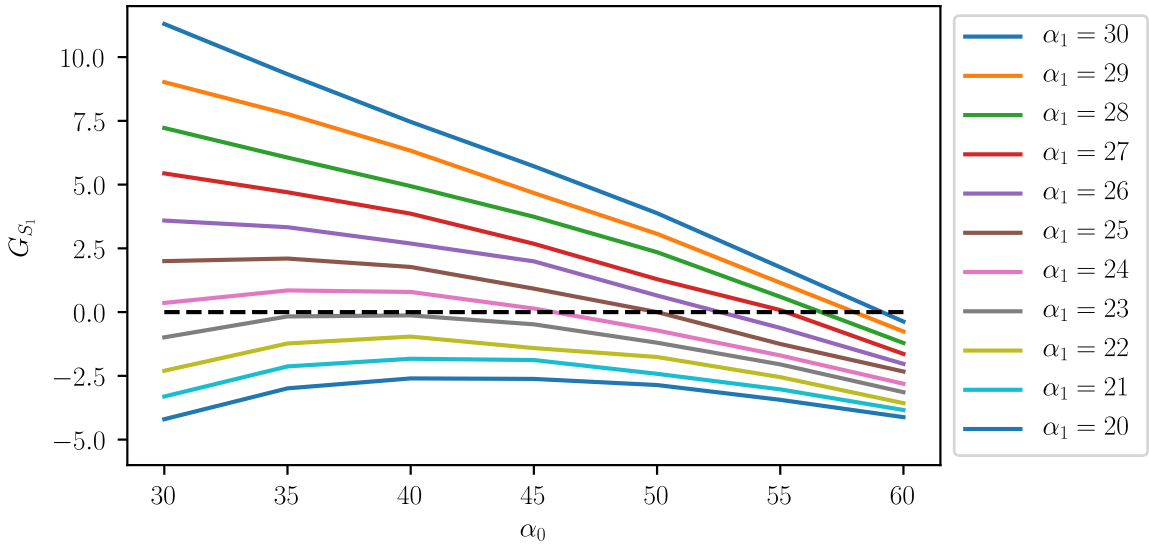


Figure 4.14: The gain for different values of α_1 , with respect to α_0

First, the weaker selfish miner will suffer from a loss in almost every scenario. There are just a few scenarios in which she will gain more revenue than her fair share, and those are the scenarios that she is pretty powerful and the difference between her power and her opponent’s power is very low. Second, the stronger selfish miner is not going to gain more revenue in every situation. Instead, she will only benefit from selfish mining if α_t is approximately greater than 0.5 (50%). This means, when the combined power of two selfish miners is at least half of the network’s hash power, it is profitable for the stronger selfish miner to continue mining selfishly. The results of the case with three selfish miners show that this conclusion is also valid for the three selfish miners, meaning that when selfish miners have the majority of the hash rate combined, selfish mining is going to be profitable, but just for the strongest one, not all of them. It means the wiser choice for the weaker selfish miners is to join the honest miners to be able to neutralize the effect of selfish mining for the strongest selfish miner.

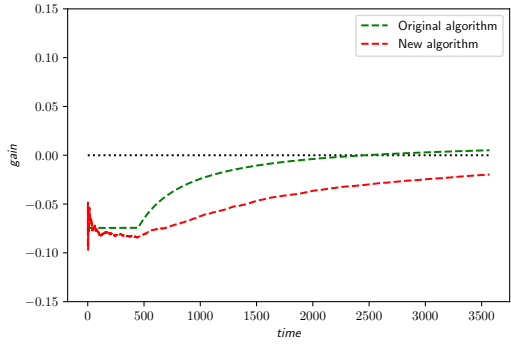
4.3.3 Zeno’s DAA

Now we can take a look at the results using our proposed DAA, *Zeno’s DAA*. We implemented our proposed Zeno’s DAA within the simulator to see its impact on the gain of selfish miners. Our difficulty adjustment algorithm’s final goal is to postpone the break-even point, which will extend the period that selfish mining is non-profitable for the selfish miner.

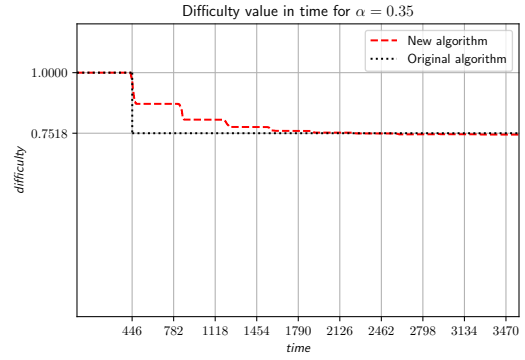
4.3.3.1 Discussion on Q_1

To answer the the first research question, Q_1 , we analyze the gain for selfish mining in different configurations. Figures 4.15, 4.16 and 4.17 show the gain and also the difficulty for networks configurations of $\alpha = 0.35$, $\alpha = 0.40$, and $\alpha = 0.45$, respectively. In each figure, subfigure (a) shows the amount of gain and subfigure (b) shows

the amount of difficulty with respect to time.

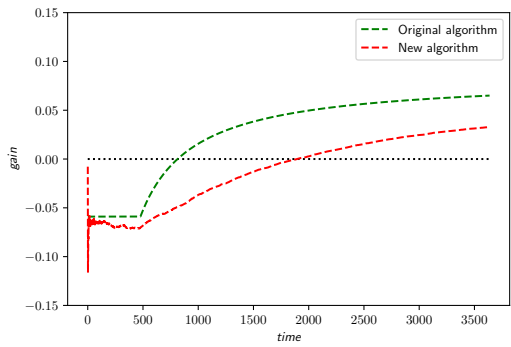


(a) Selfish miner gain

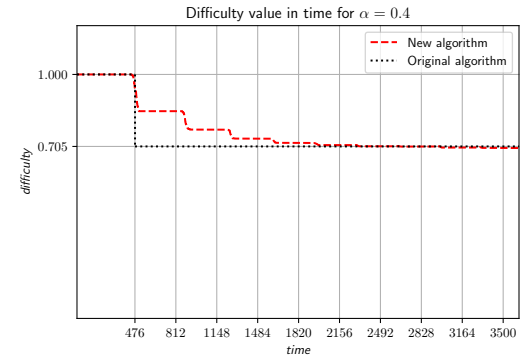


(b) Network difficulty

Figure 4.15: Comparison of different DAAs ($\alpha = 0.35$)

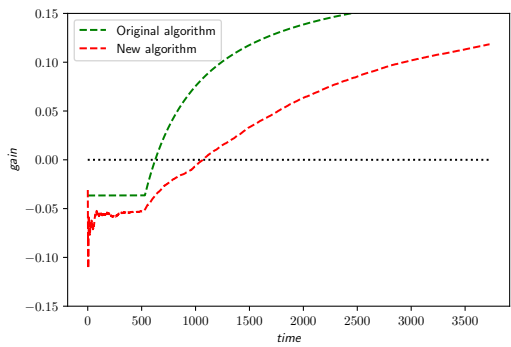


(a) Selfish miner gain

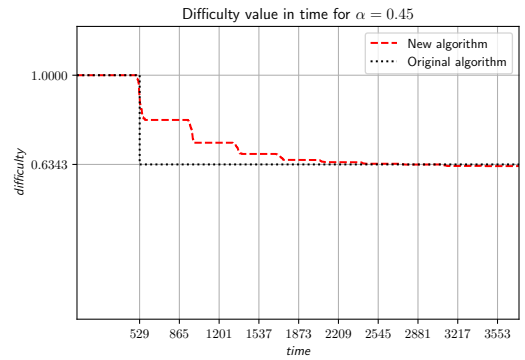


(b) Network difficulty

Figure 4.16: Comparison of different DAAs ($\alpha = 0.40$)



(a) Selfish miner gain



(b) Network difficulty

Figure 4.17: Comparison of different DAAs ($\alpha = 0.45$)

As seen in Figures 4.15a, 4.16a and 4.17a, the new algorithm decreases the slopes of the gain curves for the selfish miner, and also the break-even times (which is the point where the curves cross the black line) goes further in time, which makes the non-profitable period of selfish mining longer.

To be able to better understand the effect of Zeno’s DAA over a larger range of α values, we summarize the results in Figure 4.18. This figure shows the average break-even time for a range of α . In this figure, $\gamma = 0$; Therefore, based on our simulations, selfish mining only becomes profitable when $0.36 < \alpha$, and that is the range we included in this figure. As seen in this figure, for all values of α , Zeno’s DAA increases the break-even time, and in many cases doubles it. This is desirable for us because our goal is to discourage selfish miners by extending the period that selfish mining is non-profitable for the selfish miners so that they avoid this strategy. Although, with α increasing (meaning when the selfish miner becomes so strong that it almost has the majority), the effect of Zeno’s DAA diminishes and the amount of increase in break-even time is less.

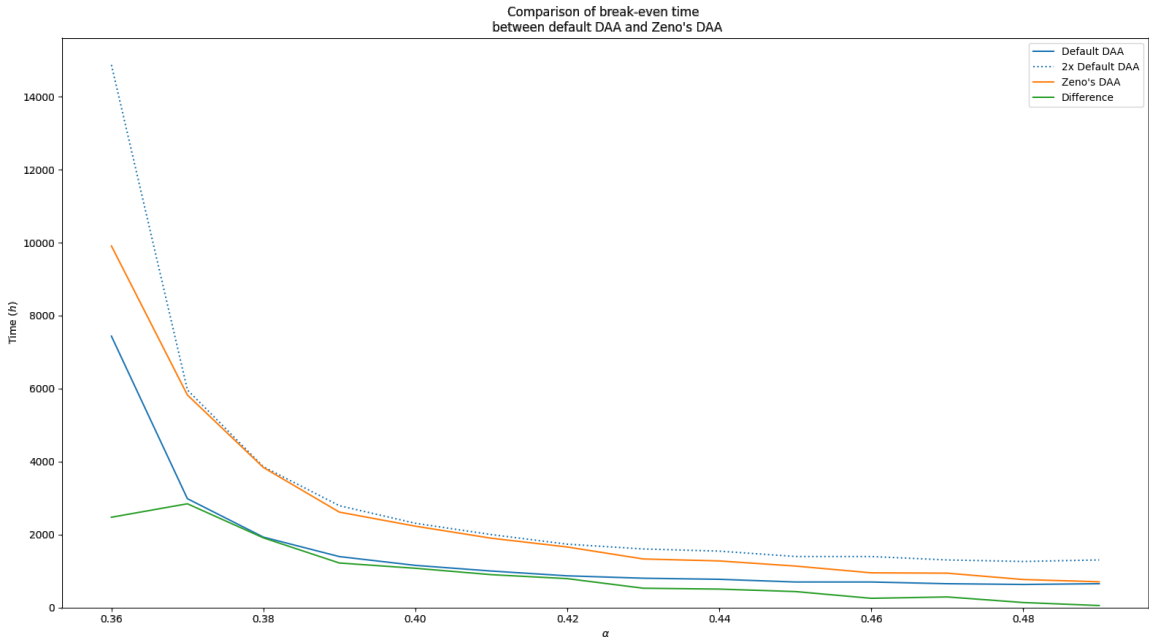


Figure 4.18: Comparison of break-even times between default DAA and Zeno’s DAA over a range of α

In Figures 4.15b, 4.16b and 4.17b, one can see that in the new difficulty adjustment algorithm, the amount of difficulty decreases gradually in steps, despite the original algorithm that changes the difficulty suddenly and in one step.

The analysis above covers Q_1 from our research questions in Section 4.1 which is related to P_1 from the list of properties of an alternative DAA that is mentioned in Section 3.2 (Page 46).

4.3.3.2 Discussion on Q_2

To answer the second question, Q_2 , we have to analyze the block generation rate of the network. As discussed before, a selfish miner slows the whole network down by performing the attack and waits for the difficulty adjustment to adjust the difficulty so that the block generation time goes back to the desired value. The default DAA does this suddenly, and the block generation time goes back to normal at the earliest possible. However, the whole idea of our proposed DAA is based on a gradual alteration of difficulty so that selfish miners' profitability waiting time becomes longer and longer and thus, selfish miners get discouraged from performing the attack. Therefore, increasing the break-even point contradicts constant block generation time. This is because of the trilemma that we discuss in Section 1.1.5. This means by using Zeno's DAA, we improve the security of the network, but it decreases the scalability of the network. So, with our proposed DAA, we cannot achieve the best block generation time, but we try to keep it as close as possible to the desired value. Figure 4.24 depicts a comparison between the two algorithms: Default DAA and our proposed Zeno's DAA. To understand the situation before any selfish mining attack, in this scenario, selfish mining starts the attack not from the beginning but right after the first difficulty adjustment (showed in the green vertical line). Figure 4.24a shows the comparison of the difficulty values for the two DAAs. This figure is similar to Figure 4.15b. Figure 4.24b shows the average block generation time for

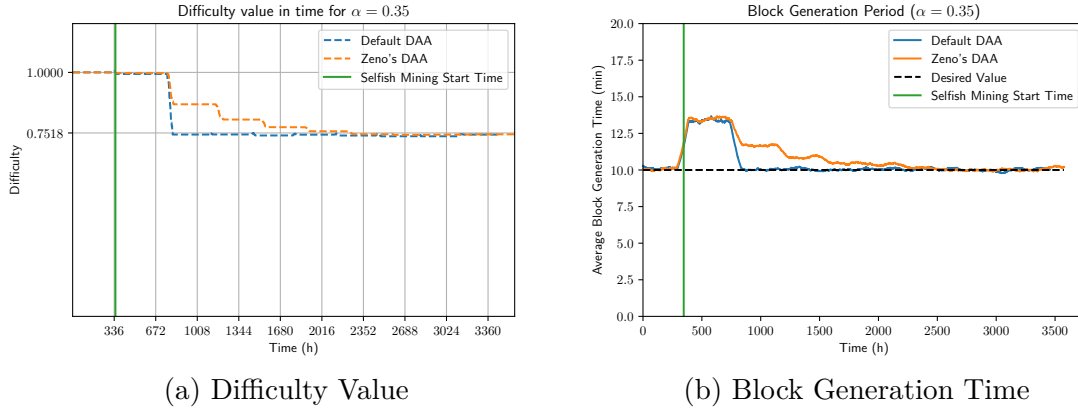


Figure 4.19: Comparison between two DAAs

the same scenario. Before the selfish mining attack, the average block generation times for both algorithms are at the desired value of ten minutes. After the first difficulty adjustment, the attacker starts its attack and suddenly, block generation time for both algorithms increase as expected. However, after the second difficulty adjustment, the default algorithm suddenly changes the average block generation time to ten minutes by lowering the difficulty. But our proposed Zeno's DAA does this gradually in multiple steps. Although the average block generation time for Zeno's DAA is not as ideal as the default DAA, we argue that its effect on the average block generation time is not severe and can be tolerated. At the same time, it tries to achieve its ultimate goal, which is preventing selfish mining.

This analysis shows that our new proposed method (Zeno's DAA) could satisfy the first characteristic of a suitable DAA and keep the second characteristic within a tolerable range. So, Zeno's DAA is in the right direction toward the ultimate goal.

4.3.3.3 The Effect Zeno's DAA on a Network without an Attacker

Zeno's DAA, just like any other DAA, adjusts the difficulty value in case of a change (either increase or decrease) in the average block generation time. The cause of this change is irrelevant to the DAA whether it is because of an attack or it is just the usual fluctuations in the total hashpower of the network. Usually, when there are

no attackers in the network, the changes in the total hashpower of the network are minor and not drastic. But what happens if we have a drastic change in the network? Assume that at one point, a group of miners with 20% of the network's hashpower leave the network. How long does it take for the difficulty value to reach the goal (and consequently, the block generation rate to get back to the desired value)? We calculated Equation (3.19) to answer this question.

In this case, with a 20% decrease in the hashpower, X would be 0.8. For the value of δ , one standard deviation would be a rational and conservative value. Based on our experiments, the standard deviation for the length of a period is about 2-3% of its length. For example, in Equation (4.2), $\sigma \approx 8$, which is roughly 2.38% of the length of a period. So, let's choose the value of 0.025 (or 2.5%) for δ . Then we will have:

$$\begin{aligned} & \log_2 \left(\frac{1}{\delta} \times \left| \frac{1-X}{X} \right| \right) < n \\ \Rightarrow & \log_2 \left(\frac{1}{0.025} \times \left| \frac{1-0.8}{0.8} \right| \right) < n \\ \Rightarrow & 3.32 < n \quad \Rightarrow \quad n = 4 \end{aligned}$$

It means in this case it takes four periods for the difficulty value to reach the goal value. If we choose a less conservative value for the δ , e.g. the three standard deviations instead of one, we would have:

$$\begin{aligned} & \log_2 \left(\frac{1}{3 \times 0.025} \times \left| \frac{1-0.8}{0.8} \right| \right) < n \\ \Rightarrow & 1.73 < n \quad \Rightarrow \quad n = 2 \end{aligned}$$

Therefore, in just two periods, the difficulty would be within the $\pm 3\sigma$ range. Figure 4.20 shows the simulation results for the same scenario. As shown in this figure, the simulation results match the mathematical analysis: after two periods, the difficulty value is within the $\pm 3\sigma$ range and after four periods, it is within the $\pm \sigma$ range.

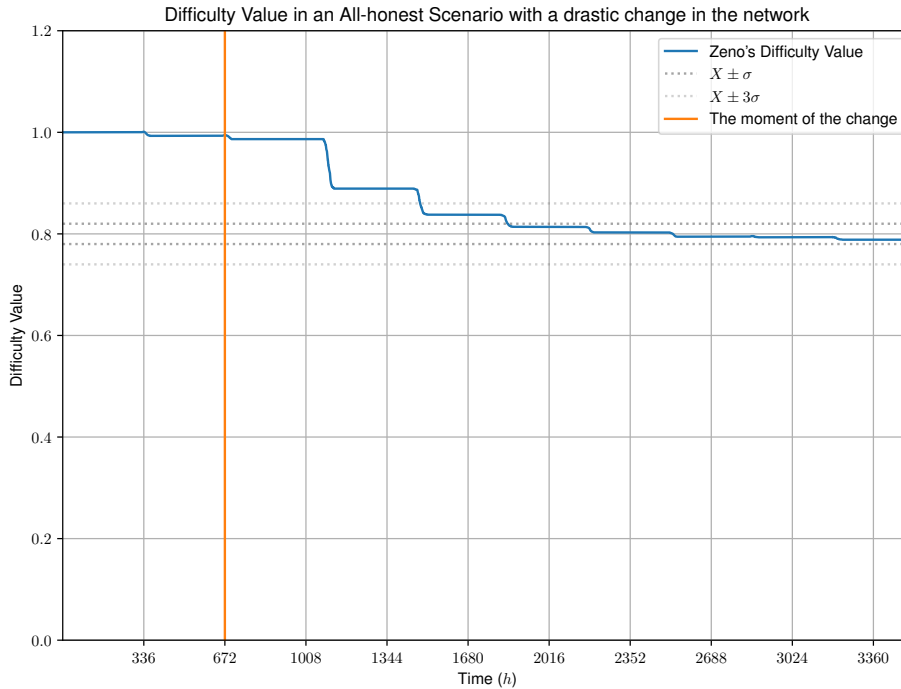
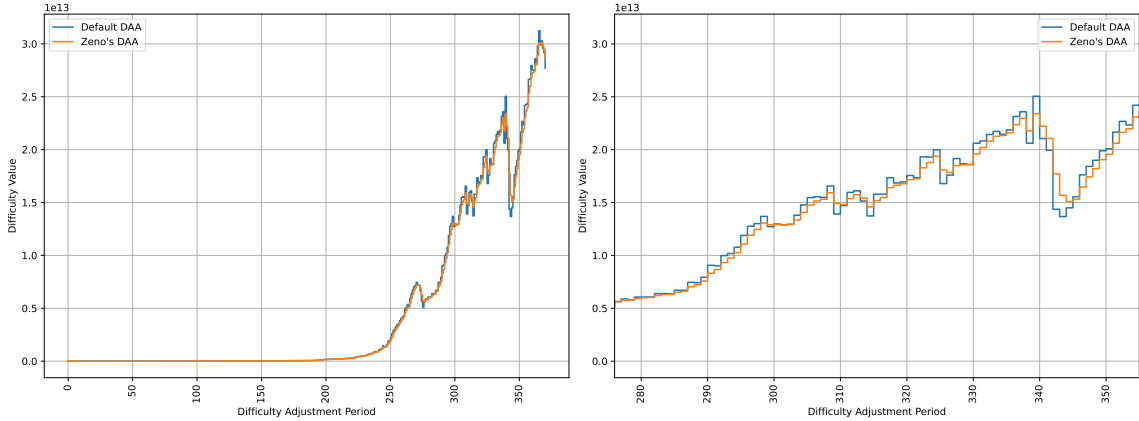


Figure 4.20: Zeno’s DAA decrease in difficulty value when there is a 20% decrease in the network’s hashpower.

4.3.3.4 Zeno’s DAA in Real World

In this section, we will explore the question of what would happen to the network if Zeno’s DAA was used instead of the default Bitcoin’s DAA. This is not an easy question to answer because there are many parameters involved, such as the effect of the DAA on block generation rate and the response of the network to the DAA. Given the history of the difficulty in Bitcoin that we showed in Figure 1.1, we try to simulate the outcome of using Zeno’s instead of Bitcoin’s DAA. Figure 4.21 shows a block-by-block comparison of Zeno’s DAA and Bitcoin’s default DAA. As shown in this figure, the changes in Zeno’s DAA are smaller and more moderate than the ones from the default DAA. However, this is not a realistic comparison, especially regarding the time. Because the value of difficulty directly affects the time. For example, in case of an increase in the network’s hashpower, the difficulty should also

increase. But as stated before, Zeno’s DAA increases the difficulty gradually. So, the next period’s difficulty will be lower than the expected value, which results in a shorter period.



(a) The whole history.

(b) A shorter time frame.

Figure 4.21: Block-by-block comparison of Bitcoin’s DAA vs. Zeno’s DAA

If we take the effect of difficulty on the lengths of the periods into consideration, the outcome will look similar to what is shown in Figure 4.22. In the history of Bitcoin, there are by far more instances of increasing the difficulty rather than decreasing it. Because of this, in Zeno’s DAA, the lengths of the periods would have been usually shorter than the default DAA. Therefore, as shown in this figure, the changes in the difficulty happen sooner. However, this is also not a realistic comparison, because it does not consider the historical changes of the network’s hashpower. For example, the huge drop in the difficulty value from the figure happened in the middle of 2021. But the results of Zeno’s DAA show that it happened earlier and in late 2020, which contradicts the historical changes in the network.

To overcome this problem, we use a simple linear interpolation to estimate the network’s hash power at every moment. Then used this value to calculate the values of the difficulty and also the length of each period using Zeno’s DAA. Figure 4.23 shows the comparison of Zeno’s DAA and the default DAA in this scenario. This is a more realistic result because we considered both the changes in the period length

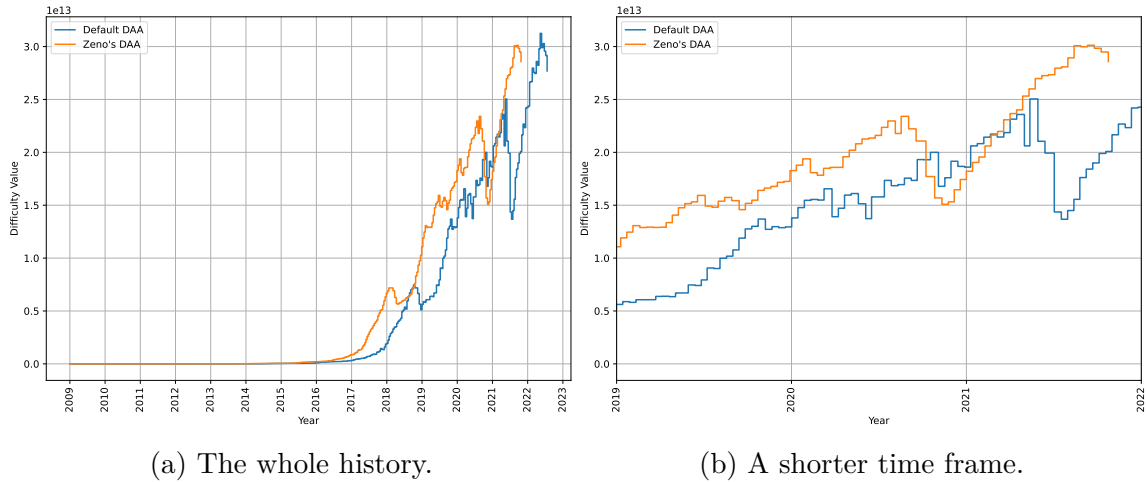


Figure 4.22: Comparison of Bitcoin's DAA vs. Zeno's DAA concerning changes in the periods' lengths.

and also the network's historical changes. As shown in this figure, there is a slight shift to the right for Zeno's which means that Zeno's is a little less responsive to the changes, as expected, because we try to adjust the difficulty gradually and in smaller steps. By looking at this figure, it can be concluded that even if we have used Zeno's DAA instead of the default DAA, the network would have scaled virtually the same and it would have had no negative effect on the network.

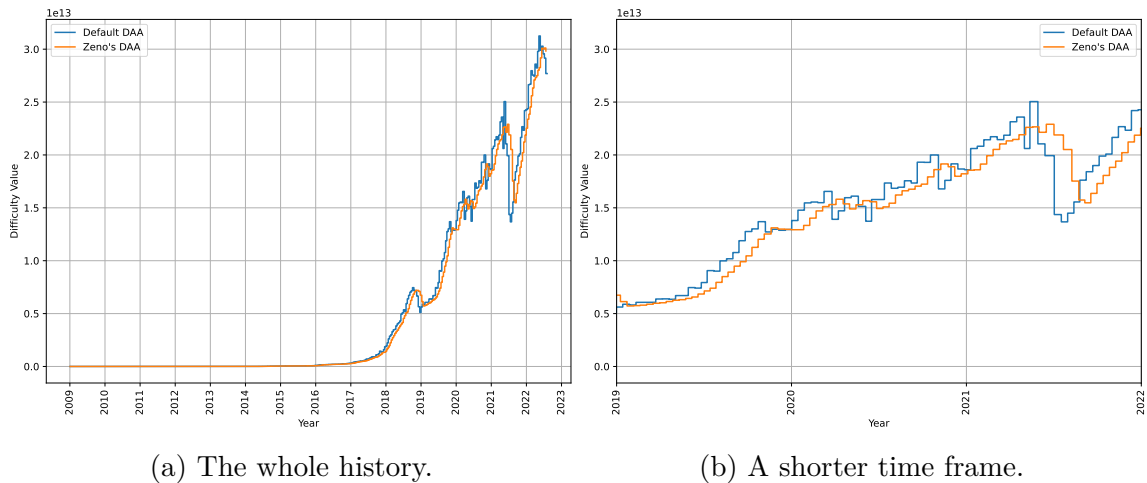


Figure 4.23: Comparison of Bitcoin's DAA vs Zeno's DAA by using an estimation for network's hash power.

4.3.4 Zeno's Max DAA

Zeno's Max DAA is only different to Zeno's DAA in increasing the difficulty. Therefore, we have to define several scenarios in which the difficulty have to be increases. For our first scenario, imagine a situation that after one period (right at the beginning of the second period), the network grows by 50%. This means that the network difficulty has to be increased by the DAA. Figure 4.24a shows the difficulty value of the network for the default DAA, Zeno's DAA and Zeno's Max DAA. As can be seen there, at the beginning of the episode (left side of the figure), default DAA and Zeno's Max behave precisely the same and do this increase in one step. However, Zeno's DAA does this in multiple phases, and it takes a lot longer to reach the desired value.

The second event in this scenario is that one miner changes its strategy from honest mining to selfish mining at the beginning of the seventh period. Because of this, the value of difficulty has to decrease to compensate for all those discarded blocks. However, as we discussed before, we want this process to be gradual to extend the waiting time of the selfish mining attackers. Therefore, as it can be seen on the right side of Figure 4.24a, this time Zeno's Max behaves similar to Zeno's instead and does this in multiple steps.

Figure 4.24b shows the average block generation time, and as it can also be seen here, for increasing the block generation time, Zeno's Max acts like the default DAA, and for decreasing it, Zeno's Max act like Zeno's DAA. This make scaling the network faster while extending the waiting time for the selfish miners.

Now let's take a look at four different scenarios and compare the performance of these three DAAs.

(S_1) Starts with honest miners; Another honest miner joins after five periods.

(S_2) Starts with honest miners; A selfish miner joins after five periods.

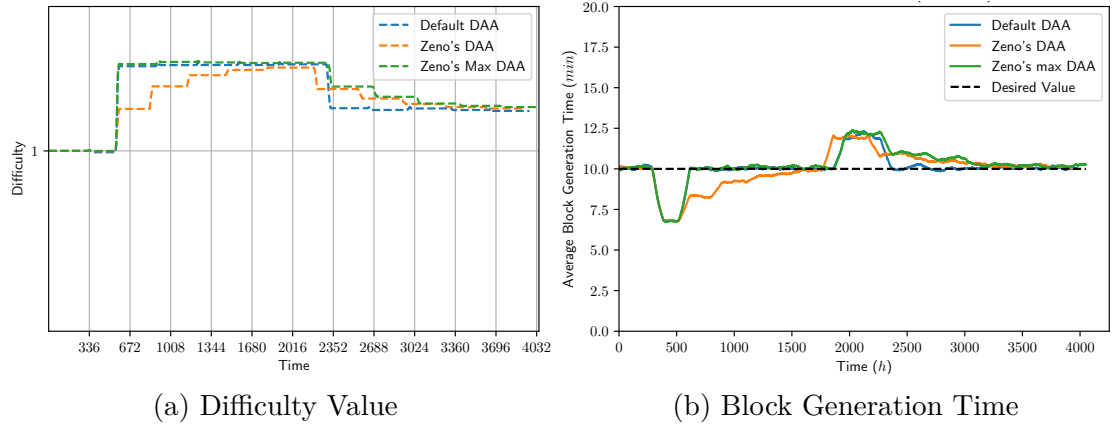


Figure 4.24: Comparison between three DAAs

(S_3) Starts with a selfish miner in the network; An honest miner joins after five periods.

(S_4) Starts with a selfish miner in the network; Another selfish miner joins after five periods.

Figure 4.25a and Figure 4.25b show the difficulty value and block generation time for the S_1 respectively. It is clear in this scenario that when we need an increase in the value of the difficulty (which directly leads to an increase to the average block generation rate), Zeno's Max and the default DAA behave similarly and do this in one step, in contrast with the Zeno's DAA that does it step by step.

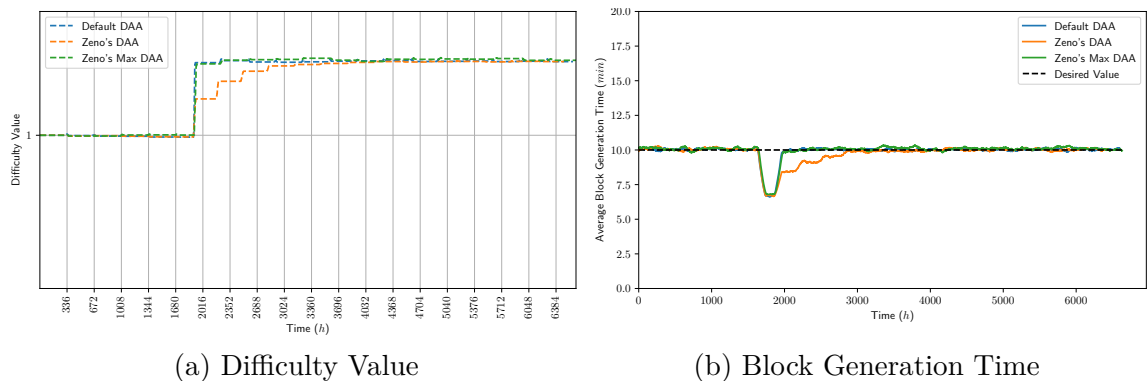


Figure 4.25: Comparison between three DAAs in S_1

In Section 3.5.7 we talked about the impact of Zeno's DAA family on honest mining

and its average block generation time. Notice that in the first five periods that there are only honest miners and no changes in the network, even with averaging between multiple simulations, all three DAA's show random fluctuations in average block generation time, and it is totally natural. This is also the case after all three DAA's bring average block generation time back to the desired value, and after that, we again have those fluctuations. This shows that in an all-honest case with small changes in the network's hash power, Zeno's DAA family does not show any noticeable difference with the default DAA. Note that here we only talk about before and long after the drastic change that we introduced in this scenario (network growth by 50%).

The second scenario is similar to the first one, but the miner who joins at the fifth period is selfish. Because we have a new miner at period five, the difficulty value has to increase, but because the new is a selfish miner, we will also have a lot of discarded blocks. Therefore, the increase in the difficulty value would not be as much as the first scenario. This can be clearly seen on Figure 4.26a in comparison to Figure 4.25a. Also, for the average block generation rate, this argument is still valid and can be seen on Figure 4.26b. Again it is apparent that in increasing the difficulty value, Zeno's Max is similar to the default DAA rather than Zeno's DAA.

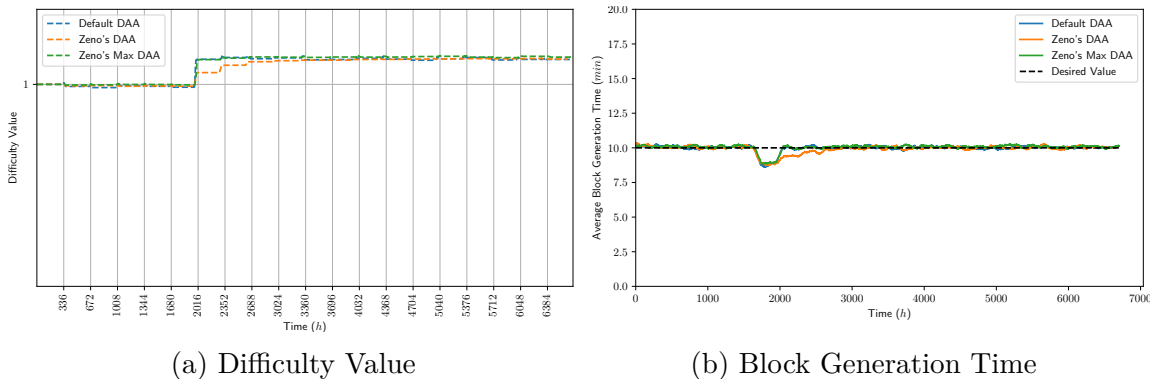


Figure 4.26: Comparison between three DAAs in S_2

The following two scenarios, S_3 and S_4 , are a bit different from the first two. In

these scenarios, we start with a selfish miner. Therefore, in the beginning, the block generation time increases (Figure 4.27b and Figure 4.28b), and we have to decrease the difficulty to take the block generation time back to its desired value. As shown at the beginning of Figure 4.27a and Figure 4.28a, the difficulty value will decrease after the first period. However, while reducing the difficulty value, Zeno's Max behaves similarly to Zeno's DAA and gradually reduces the difficulty value. But when the new miner joins the network after period five, it again acts as the default DAA. The difference between S_3 and S_4 is that in S_4 , the new miner is also selfish. Because of that, the increment in the difficulty value after period five is not as much as S_3 .

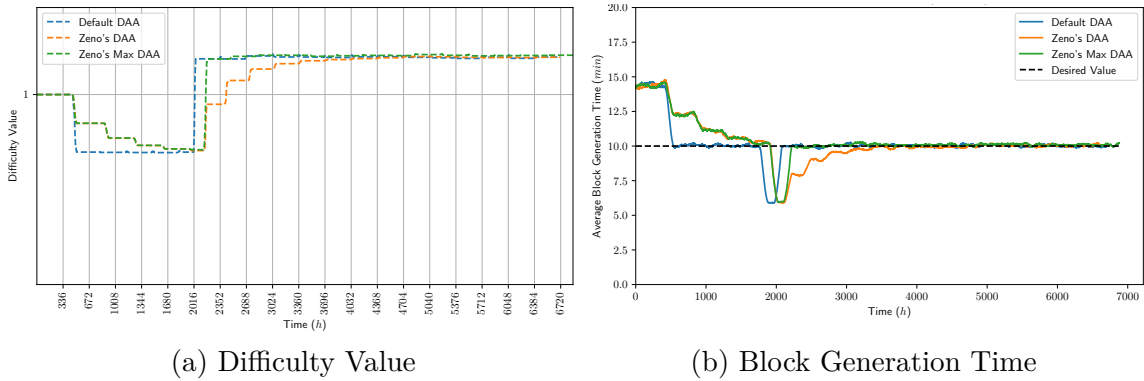


Figure 4.27: Comparison between three DAAs in S_3

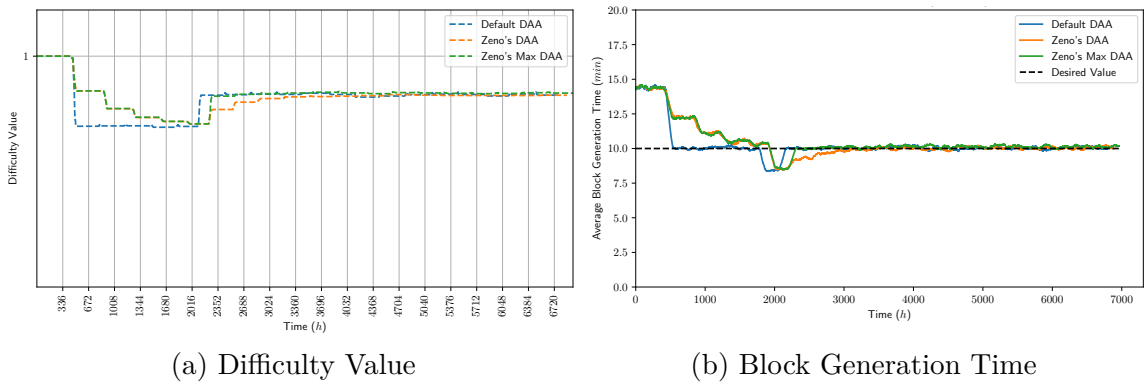


Figure 4.28: Comparison between three DAAs in S_4

4.3.5 Zeno's Parametric DAA

For evaluating this method, we take a look at the behaviour of different values for τ in the same scenario. Figure 4.29 shows the results of $G(t)$ (gain for the selfish miner) when $\alpha = 0.5$ for different values of τ . It is evident in this figure that for the larger values of τ , the break-even point (the point where the G curve reaches the value of zero) will be delayed more and more.

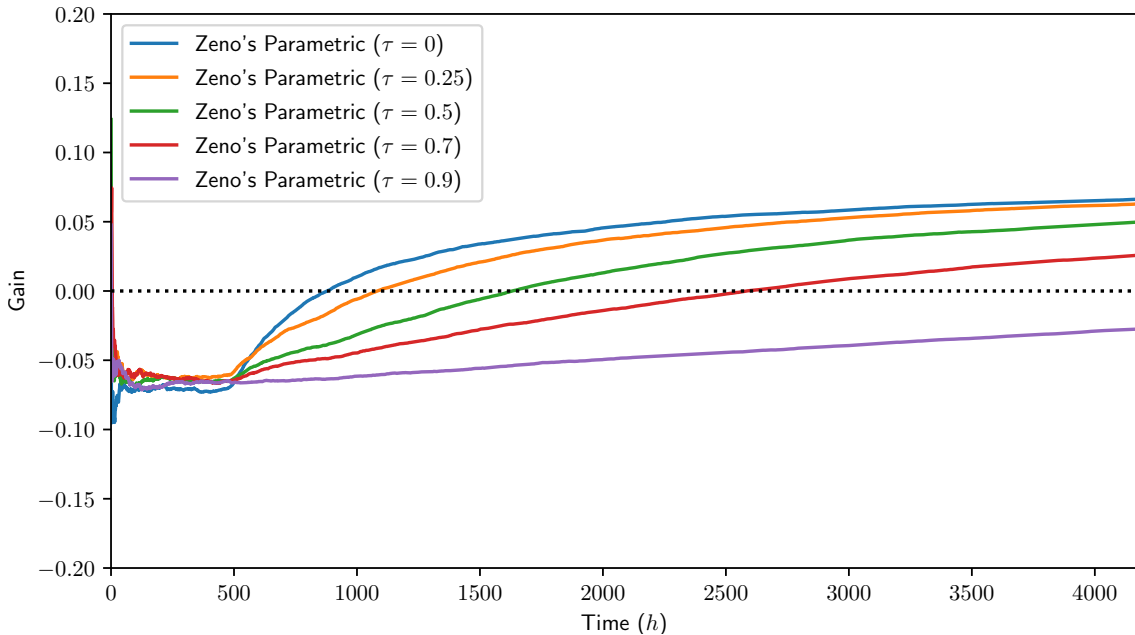


Figure 4.29: Gain for a selfish miner with $\alpha = 0.4$ for different values of τ ($\tau = 0$: Default DAA, $\tau = 0.5$: Zeno's DAA)

Figure 4.30 shows the average block generation time for different values of τ in the same scenario as above. It shows that for larger values of τ , the average block generation time will take longer to return to its intended value (ten minutes). This is the trade-off that we discuss in the previous section. If a blockchain can tolerate more prolonged periods of higher block generation time, it can use a higher value for τ to discourage selfish behaviour more intensively.

In Figure 4.31 the values for difficulties with different values of τ are depicted. As it can be seen here, difficulty values decrease step by step. However, by increasing

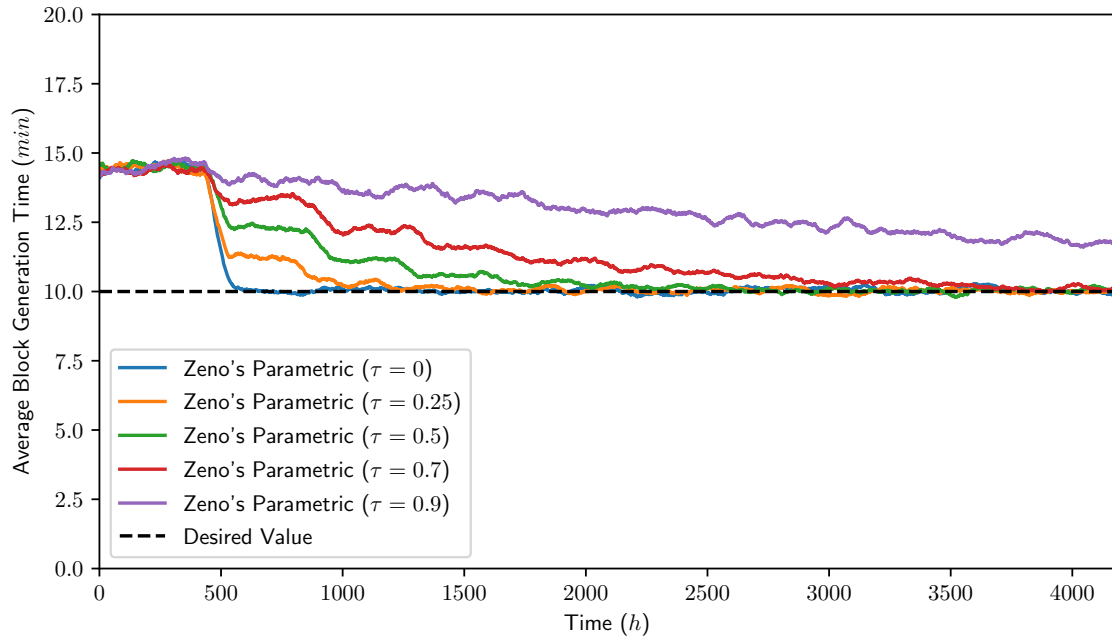


Figure 4.30: Block generation time ($\alpha = 0.4$) for different values of τ ($\tau = 0$: Default DAA, $\tau = 0.5$: Zeno's DAA)

the value of τ , the heights of the steps decrease, and the lengths of them increase.

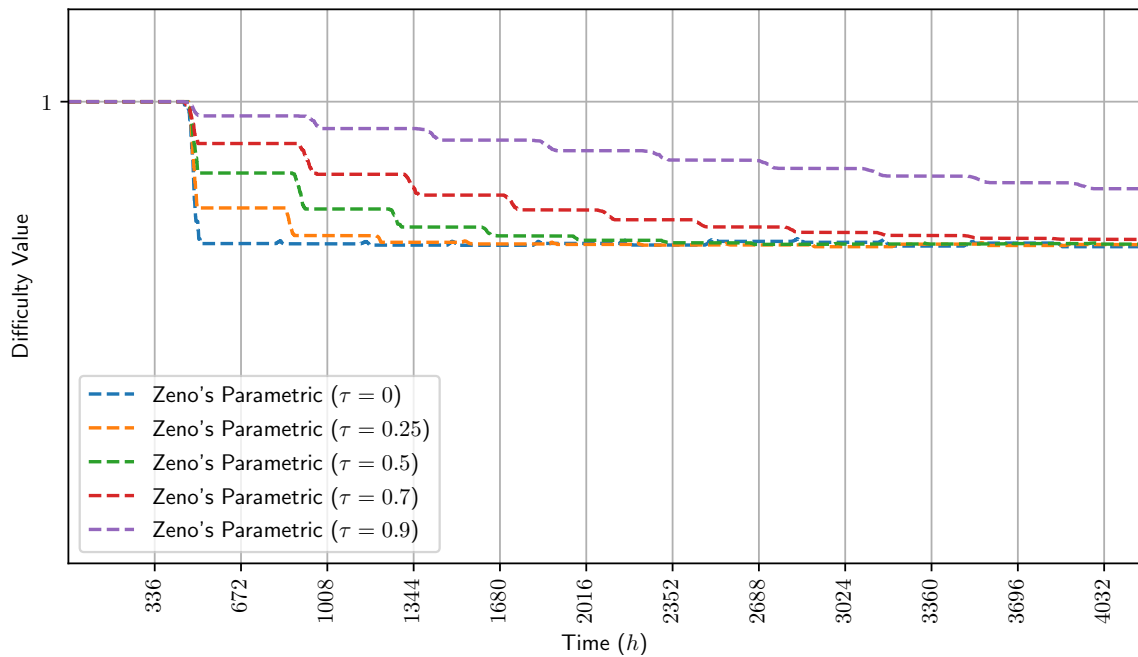


Figure 4.31: Difficulty value ($\alpha = 0.4$) for different values of τ ($\tau = 0$: Default DAA, $\tau = 0.5$: Zeno's DAA)

4.4 Summary

In this chapter, we evaluated our three proposed methods Zeno's DAA, Zeno's Max DAA, and Zeno's Parametric DAA. We used a simulation approaches for evaluation to back up the mathematical aanalysis we provided in 3.4. We presented the results of evaluations for various scenarios like having multiple attackers and using alternative DAAs. We showed that our proposed methods could effectively extend the break-even time for selfish miners to discourage them from performing the attack.

Chapter 5

Conclusion and Future Work

In this thesis, we focused on one of the most famous attacks on the mining process of Proof-of-Work blockchains, Bitcoin, in particular, called *selfish mining*. As discussed before, selfish mining works by not publishing discovered blocks right away and keeping them private as long as possible. By using such a strategy in publishing blocks, the hash power of other miners in the network would go to waste as they do not have access to the latest block that is generated by the selfish miner. Of course, as a result, a part of the hash power of the selfish miner would also go to waste. However, assuming that the selfish miner is strong enough, the amount of waste for the selfish miner would be proportionally less than the amount of waste for other miners.

Here is where difficulty adjustment comes to help the selfish miner. This considerable amount of waste in hash power will result in a lower block generation rate. Therefore, the difficulty adjustment algorithm lowers the difficulty to bring it back to the desired value. As a result, the selfish miner will get more revenue than its fair share. However, it takes some time for the selfish miner to go from an initial drop in revenue (due to the initial hash rate waste) to benefiting from selfish mining.

In this research, first, we extensively analyzed the profitability of selfish mining for

different scenarios to reach a conclusion and determine under what circumstances the selfish mining becomes profitable. We showed that time plays an important factor in the profitability of selfish mining and even the strongest selfish miners have to wait for a certain period of time before they can benefit from selfish mining and gain a revenue more than their fair share. This was the first contribution that we addressed in Section 3.1.4.

We also conducted a series of analyses on the minimum hash-power required for selfish mining, the waiting time required for selfish mining to become profitable, and more. Another analysis that we performed was about the profitability of selfish mining with multiple attackers in the networks. Imagine a case where two or more miners or mining pools decide to perform selfish mining separately without collaborating with each other. We showed that in this case, the weaker selfish miner (meaning the one with less hash power) will suffer from a loss almost all the time, and instead helps the stronger selfish miner to gain more revenue. So, for the weaker miner, performing selfish mining is not only harmful in terms of revenue but also helps the competitor and encourages it to continue the attack. This was the second contribution that we discussed in Section 4.3.2.

After defining the problem with the current difficulty adjustment algorithm that makes selfish mining profitable, we proposed an alternative difficulty adjustment algorithm, Zeno's DAA, to mitigate the issue. The main idea behind our proposed algorithm is to adjust the difficulty value gradually by considering the history of difficulty values. Zeno's DAA sets the difficulty value halfway between the current value and the goal value (which is the expected value of difficulty). The motivation behind this idea is that this will extend the waiting time for selfish miners to profit from the attack. Given the dynamicity and ever-changing nature of the Bitcoin network especially concerning its price, this extended waiting period is a strong discouraging factor. We defined our proposed algorithm in detail in Section 3.3 as

our third contribution.

We discussed that Zeno's DAA could be improved. One of the improvements that can be done to Zeno's DAA is to make it asymmetric. It means we modify it to treat increasing and decreasing the difficulty differently. That is because in increasing the difficulty, we don't need a gradual change. After all, we are not trying to extend the waiting time. Instead, we can increase the difficulty value in just one step. This created Zeno's Max, which offers better overall block generation time. Also, another improvement was to create a way that Zeno's DAA could be tuned. Because Zeno's DAA offers security with a cost of a slight increase in average block generation time. It would be a good idea to be able to tune this security and scalability trade-off. So, we introduced Zeno's Parametric DAA to implement this idea. In Zeno's Parametric DAA we have a parameter, τ . By increasing the value of τ , we increase the security, but we have to endure longer periods of higher average block generation time and vice versa. So, its value can be decided based on the needs of the blockchain. These two new algorithms extend our proposed method to a family of alternative difficulty adjustment algorithms as our fourth contribution that we covered in Chapter 3.

To show the effectiveness of our algorithms, we had to analyse and evaluate them. To do so, we used two methods of evaluation: mathematical analysis and simulation analysis. For mathematical analysis, similar to what we did to analyse the profitability of selfish mining, we used mathematical formulations of the proposed algorithms and performed the similar analysis. This was our fifth contribution and we covered it in Section 3.4.

The second evaluation method was simulation analysis. For this, we designed and developed a Bitcoin mining simulator capable of simulating different difficulty adjustment algorithms. We talked about this simulator in Section 4.2 and also Appendix A. We used it to simulate different scenarios and compare them to each other. This was our sixth contribution that we covered in Sections 4.3.3 to 4.3.5. By comparing

the waiting time of the selfish miner for two scenarios, one using Bitcoin’s default DAA and the other one using Zeno’s DAA, we showed that we could effectively increase the waiting time to almost double in many configurations and discourage selfish behaviour.

5.1 Future Works

There are a few line of research that could be considered to continue and complement this work. Here is a list of them:

- Our proposed method is designed to take the history of difficulty value into consideration instead of only using the expected difficulty of the last period. It would be a good idea to explore other ways to implement this idea. For example using the real and expected difficulty values of multiple periods might provide interesting results.
- As we mentioned in Section 3.5.3, one interesting line of research could be extending Zeno’s Parametric to be a dynamic DAA. Zeno’s Parametric is designed to work with a constant value of τ for the entire lifetime of the blockchain. However, this value could be a dynamic value that can be changed based on some criteria or an algorithm. However, it need careful consideration, because for all the peer should be able to calculate the new value independently.
- There are a few other types of difficulty adjustment algorithms used in other blockchains and altcoins like BSV. However, these DAAs focus more on scalability (keeping block generation rate constant and making difficulty adjustment more responsive). Therefore, based on our argument about trade-off between scalability, they should by definition be weaker against selfish mining attack. A comparative analysis on these alternative DAA would be another future line of research for this thesis.

- In chapter 2, we discussed a few variant of selfish mining and other attacks that are similar. Also, we discussed that a proper alternative DAA is ideally resistant to other types of attacks based on difficulty manipulation. So, there is a potential line of research to analyze the effectiveness of the proposed DAAs against those attacks.
- Another line of potential works could be about the implementation of our simulator. There are studies that combine selfish mining and other types of attacks like eclipse attacks that we discussed in Chapter 2. Our simulator is currently unable to simulate eclipse attack. Because this attack works based on the peer-to-peer structure of the network. So, it would be a good idea to develop the simulator to be able to simulate this attack.

References

- [1] *Difficulty (bitcoin wiki)*, 2021, [Online; Accessed May 2022].
- [2] Elli Androulaki, Ghassan O Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun, *Evaluating user privacy in bitcoin*, International Conference on Financial Cryptography and Data Security, Springer, 2013, pp. 34–51.
- [3] Hamid Azimy and Ali Ghorbani, *Competitive selfish mining*, 2019 17th International Conference on Privacy, Security and Trust (PST), IEEE, 2019, pp. 1–8.
- [4] Samiran Bag, Sushmita Ruj, and Kouichi Sakurai, *Bitcoin block withholding attack: Analysis and mitigation*, IEEE Transactions on Information Forensics and Security **12** (2016), no. 8, 1967–1978.
- [5] Samiran Bag and Kouichi Sakurai, *Yet another note on block withholding attack on bitcoin mining pools*, International Conference on Information Security, Springer, 2016, pp. 167–180.
- [6] Lear Bahack, *Theoretical bitcoin attacks with less than half of the computational power (draft)*, arXiv preprint arXiv:1312.7013 (2013).
- [7] S Banupriya and K Kottilingam, *An analysis of privacy issues and solutions in public blockchain (bitcoin)*, 2021 2nd International Conference for Emerging Technology (INCET), IEEE, 2021, pp. 1–7.

- [8] Karolina Bergman and Saeed Rajput, *Revealing and concealing bitcoin identities: A survey of techniques*, Proceedings of the 3rd ACM International Symposium on Blockchain and Secure Critical Infrastructure, 2021, pp. 13–24.
- [9] Jorge Bernal Bernabe, Jose Luis Canovas, Jose L Hernandez-Ramos, Rafael Torres Moreno, and Antonio Skarmeta, *Privacy-preserving solutions for blockchain: Review and challenges*, IEEE Access **7** (2019), 164908–164940.
- [10] George Bissias, A Pinar Ozisik, Brian N Levine, and Marc Liberatore, *Sybil-resistant mixing for bitcoin*, Proceedings of the 13th Workshop on Privacy in the Electronic Society, 2014, pp. 149–158.
- [11] Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, Joshua A Kroll, and Edward W Felten, *Mixcoin: Anonymity for bitcoin with accountable mixes*, International Conference on Financial Cryptography and Data Security, Springer, 2014, pp. 486–504.
- [12] Rafael Brune, *Bitcoin network simulator*, 2013, [Online; Accessed Jul. 2020].
- [13] Vanessa Chicarino, Célio Albuquerque, Emanuel Jesus, and Antônio Rocha, *On the detection of selfish mining and stalker attacks in blockchain networks*, Annals of Telecommunications (2020), 1–10.
- [14] Mauro Conti, E Sandeep Kumar, Chhagan Lal, and Sushmita Ruj, *A survey on security and privacy issues of bitcoin*, IEEE Communications Surveys & Tutorials **20** (2018), no. 4, 3416–3452.
- [15] Nicolas T Courtois and Lear Bahack, *On subversive miner strategies and block withholding attack in bitcoin digital currency*, arXiv preprint arXiv:1402.1718 (2014).
- [16] Wei Dai, *b-money*, 1998, [Online; Accessed Jul. 2020].

- [17] Christian Decker and Roger Wattenhofer, *Information propagation in the bitcoin network*, Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on, IEEE, 2013, pp. 1–10.
- [18] Xuewen Dong, Feng Wu, Anter Faree, Deke Guo, Yulong Shen, and Jianfeng Ma, *Selfholding: A combined attack model using selfish mining with block withholding attack*, Computers & Security **87** (2019), 101584.
- [19] Evan Duffield and Daniel Diaz, *Dash: A privacycentric cryptocurrency*, 2015.
- [20] Ittay Eyal and Emin Gün Sirer, *Bitcoin is broken*, 2013, [Online; Accessed Feb. 2020].
- [21] ———, *Majority is not enough: Bitcoin mining is vulnerable*, International conference on financial cryptography and data security, Springer, 2014, pp. 436–454.
- [22] Chen Feng and Jianyu Niu, *Selfish mining in ethereum*, 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), IEEE, 2019, pp. 1306–1316.
- [23] Qi Feng, Debiao He, Sherali Zeadally, Muhammad Khurram Khan, and Neeraj Kumar, *A survey on privacy protection in blockchain system*, Journal of Network and Computer Applications **126** (2019), 45–58.
- [24] Michael Fleder, Michael S Kester, and Sudeep Pillai, *Bitcoin transaction graph analysis*, arXiv preprint arXiv:1502.01657 (2015).
- [25] Jake Frankenfield, *Selfish mining*, [Online; Accessed Jun. 2022].
- [26] Daniel Fullmer and A Stephen Morse, *Analysis of difficulty control in bitcoin and proof-of-work blockchains*, 2018 IEEE Conference on Decision and Control (CDC), IEEE, 2018, pp. 5988–5992.

- [27] I Gusti Ayu Kusdiah Gemeliarana and Riri Fitri Sari, *Evaluation of proof of work (pow) blockchains security network on selfish mining*, 2018 International Seminar on Research of Information Technology and Intelligent Systems (IS-RITI), IEEE, 2018, pp. 126–130.
- [28] Arthur Gervais, *Bitcoin simulator*, 2016, [Online; Accessed Jul. 2020].
- [29] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun, *On the security and performance of proof of work blockchains*, Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, 2016, pp. 3–16.
- [30] Johannes Göbel, Holger Paul Keeler, Anthony E Krzesinski, and Peter G Taylor, *Bitcoin blockchain dynamics: The selfish-mine strategy in the presence of propagation delay*, Performance Evaluation **104** (2016), 23–41.
- [31] Cyril Grunspan and Ricardo Pérez-Marco, *On profitability of selfish mining*, arXiv preprint arXiv:1805.08281 (2018).
- [32] ———, *Selfish mining in ethereum*, arXiv preprint arXiv:1904.13330 (2019).
- [33] Tobias Guggenberger, Vincent Schlatt, Jonathan Schmid, and Nils Urbach, *A structured overview of attacks on blockchain systems*, Proceedings of the Pacific Asia Conference on Information Systems (PACIS), 2021.
- [34] Dave Gutteridge, *Japanese cryptocurrency monacoin hit by selfish mining attack*, [Online; Accessed Jun. 2022].
- [35] Ethan Heilman, *One weird trick to stop selfish miners: Fresh bitcoins, a solution for the honest miner*, International Conference on Financial Cryptography and Data Security, Springer, 2014, pp. 161–162.

- [36] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg, *Eclipse attacks on bitcoins peer-to-peer network*, 24th {USENIX} Security Symposium ({USENIX} Security 15), 2015, pp. 129–144.
- [37] Victor Holotescu and Radu Vasiu, *Challenges and emerging solutions for public blockchains*, BRAIN. Broad Research in Artificial Intelligence and Neuroscience **11** (2020), no. 1, 58–83.
- [38] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox, *Zcash protocol specification*, GitHub: San Francisco, CA, USA (2016).
- [39] Hongyue Kang, Xiaolin Chang, Runkai Yang, Jelena Mišić, and Vojislav B Mišić, *Understanding selfish mining in imperfect bitcoin and ethereum networks with extended forks*, IEEE Transactions on Network and Service Management **18** (2021), no. 3, 3079–3091.
- [40] Ghassan Karame and Srdjan Capkun, *Blockchain security and privacy*, IEEE Security & Privacy **16** (2018), no. 04, 11–12.
- [41] Michal Kędziora, Patryk Kozłowski, Michał Szczepanik, and Piotr Józwiak, *Analysis of blockchain selfish mining attacks*, International Conference on Information Systems Architecture and Technology, Springer, 2019, pp. 231–240.
- [42] Merve Can Kus Khalilov and Albert Levi, *A survey on anonymity and privacy in bitcoin-like digital cash systems*, IEEE Communications Surveys & Tutorials **20** (2018), no. 3, 2543–2585.
- [43] Sunny King, *Primecoin: Cryptocurrency with prime number proof-of-work*, July 7th (2013).
- [44] Philip Koshy, Diana Koshy, and Patrick McDaniel, *An analysis of anonymity in bitcoin using p2p network traffic*, International Conference on Financial Cryptography and Data Security, Springer, 2014, pp. 469–485.

- [45] Yujin Kwon, Dohyun Kim, Yunmok Son, Eugene Vasserman, and Yongdae Kim, *Be selfish and avoid dilemmas: Fork after withholding (faw) attacks on bitcoin*, Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2017, pp. 195–209.
- [46] Suhyeon Lee and Seungjoo Kim, *Detective mining: Selfish mining becomes unrealistic under mining pool environment*, 2019.
- [47] Tin Leelavimolsilp, Long Tran-Thanh, and Sebastian Stein, *On the preliminary investigation of selfish mining strategy with multiple selfish miners*, arXiv preprint arXiv:1802.02218 (2018).
- [48] Tao Li, Zhaojie Wang, Guoyu Yang, Yang Cui, Yuling Chen, and Xiaomei Yu, *Semi-selfish mining based on hidden markov decision process*, International Journal of Intelligent Systems **36** (2021), no. 7, 3596–3612.
- [49] Gregory Maxwell, *Coinjoin: Bitcoin privacy for the real world*, 2013, [Online; Accessed Feb. 2021].
- [50] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage, *A fistful of bitcoins: characterizing payments among men with no names*, Proceedings of the 2013 conference on Internet measurement conference, 2013, pp. 127–140.
- [51] Dmitry Meshkov, Alexander Chepurnoy, and Marc Jansen, *Short paper: Revisiting difficulty control for blockchain systems*, Data Privacy Management, Cryptocurrencies and Blockchain Technology, Springer, 2017, pp. 429–436.
- [52] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin, *Zerocoin: Anonymous distributed e-cash from bitcoin*, Security and Privacy (SP), 2013 IEEE Symposium on, IEEE, 2013, pp. 397–411.

- [53] Bhabendu Kumar Mohanta, Debasish Jena, Soumyashree S Panda, and Srichandan Sobhanayak, *Blockchain technology: A survey on applications and security privacy challenges*, *Internet of Things* **8** (2019), 100107.
- [54] Satoshi Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, 2008.
- [55] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi, *Stubborn mining: Generalizing selfish mining and combining with an eclipse attack*, *Security and Privacy (EuroS&P)*, 2016 IEEE European Symposium on, IEEE, 2016, pp. 305–320.
- [56] Kervins Nicolas, Yi Wang, and George C Giakos, *Comprehensive overview of selfish mining and double spending attack countermeasures*, 2019 IEEE 40th Sarnoff Symposium, IEEE, 2019, pp. 1–6.
- [57] Jianyu Niu and Chen Feng, *Selfish mining in ethereum*, arXiv preprint arXiv:1901.04620 (2019).
- [58] Shunya Noda, Kyohei Okumura, and Yoshinori Hashimoto, *An economic analysis of difficulty adjustment algorithms in proof-of-work blockchain systems*, Available at SSRN 3410460 (2019).
- [59] Remigijus Paulavičius, Saulius Grigaitis, and Ernestas Filatovas, *A systematic review and empirical analysis of blockchain simulators*, IEEE access **9** (2021), 38010–38028.
- [60] Fergal Reid and Martin Harrigan, *An analysis of anonymity in the bitcoin system*, *Security and privacy in social networks*, Springer, 2013, pp. 197–223.
- [61] Fabian Ritz and Alf Zugenmaier, *The impact of uncle rewards on selfish mining in ethereum*, 2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), IEEE, 2018, pp. 50–57.

- [62] Meni Rosenfeld, *Analysis of bitcoin pooled mining reward systems*, arXiv preprint arXiv:1112.4980 (2011).
- [63] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate, *Coinshuffle: Practical decentralized coin mixing for bitcoin*, European Symposium on Research in Computer Security, Springer, 2014, pp. 345–364.
- [64] Muhammad Saad, Laurent Njilla, Charles Kamhoua, and Aziz Mohaisen, *Countering selfish mining in blockchains*, 2019 International Conference on Computing, Networking and Communications (ICNC), IEEE, 2019, pp. 360–364.
- [65] Muhammad Saad, Jeffrey Spaulding, Laurent Njilla, Charles Kamhoua, Sachin Shetty, DaeHun Nyang, and Aziz Mohaisen, *Exploring the attack surface of blockchain: A systematic overview*, arXiv preprint arXiv:1904.03487 (2019).
- [66] Muhammad Saad, Jeffrey Spaulding, Laurent Njilla, Charles Kamhoua, Sachin Shetty, DaeHun Nyang, and David Mohaisen, *Exploring the attack surface of blockchain: A comprehensive survey*, IEEE Communications Surveys & Tutorials **22** (2020), no. 3, 1977–2008.
- [67] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar, *Optimal selfish mining strategies in bitcoin*, International Conference on Financial Cryptography and Data Security, Springer, 2016, pp. 515–532.
- [68] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza, *Zerocash: Decentralized anonymous payments from bitcoin*, 2014 IEEE Symposium on Security and Privacy (SP), IEEE, 2014, pp. 459–474.
- [69] Santhi Shalini and H Santhi, *A survey on various attacks in bitcoin and cryptocurrency*, 2019 International Conference on Communication and Signal Processing (ICCSP), IEEE, 2019, pp. 0220–0224.

- [70] Siamak Solat and Maria Potop-Butucaru, *Zeroblock: Preventing selfish mining in bitcoin*, arXiv preprint arXiv:1605.02435 (2016).
- [71] Luke Valenta and Brendan Rowan, *Blindcoin: Blinded, accountable mixes for bitcoin*, International Conference on Financial Cryptography and Data Security, Springer, 2015, pp. 112–126.
- [72] Surya Viswanathan and Aakash Shah, *Scalability trilemma*, 2018, [Online; Accessed Dec. 2021].
- [73] Canhui Wang, Xiaowen Chu, and Qin Yang, *Measurement and analysis of the bitcoin networks: A view from mining pools*, arXiv preprint arXiv:1902.07549 (2019).
- [74] Heli Wang, Qiao Yan, and Victor CM Leung, *The impact of propagation delay to different selfish miners in proof-of-work blockchains*, Peer-to-Peer Networking and Applications (2021), 1–8.
- [75] Zhaojie Wang, Qingzhe Lv, Zhaobo Lu, Yilei Wang, and Shengjie Yue, *Forkdec: Accurate detection for selfish mining attacks*, Security and Communication Networks **2021** (2021).
- [76] Yujuan Wen, Fengyuan Lu, Yufei Liu, and Xinli Huang, *Attacks and countermeasures on blockchains: A survey from layering perspective*, Computer Networks **191** (2021), 107978.
- [77] Craig Wright, *Jean-paul sartre, signing and significance*, 2016, [Online; Accessed Jan. 2022].
- [78] Craig S Wright, *The fallacy of selfish mining in bitcoin: A mathematical critique*, Available at SSRN 3004026 (2017).

- [79] Craig S Wright and Stephane Savanah, *The fallacy of the selfish miner in bitcoin: An economic critique*, Available at SSRN 3009466 (2017).
- [80] Qing Xia, Wensheng Dou, Tong Xi, Jing Zeng, Fengjun Zhang, Jun Wei, and Geng Liang, *The impact analysis of multiple miners and propagation delay on selfish mining*, 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC), IEEE, 2021, pp. 694–703.
- [81] Runkai Yang, Xiaolin Chang, Jelena Mišić, and Vojislav B Mišić, *Assessing blockchain selfish mining in an imperfect network: Honest and selfish miner views*, *Computers & Security* **97** (2020), 101956.
- [82] Rui Zhang, Rui Xue, and Ling Liu, *Security and privacy on blockchain*, *ACM Computing Surveys (CSUR)* **52** (2019), no. 3, 1–34.
- [83] Shiquan Zhang, Kaiwen Zhang, and Bettina Kemme, *A simulation-based analysis of multiplayer selfish mining*, 2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), IEEE, 2020, pp. 1–5.
- [84] Wenyu Zhang, Zhenjiang Zhang, Han-Chieh Chao, and Mohsen Guizani, *Toward intelligent network optimization in wireless networking: An auto-learning framework*, *IEEE Wireless Communications* **26** (2019), no. 3, 76–82.
- [85] Wenyu Zhang, Zhenjiang Zhang, Sherali Zeadally, Han-Chieh Chao, and Victor CM Leung, *Masm: A multiple-algorithm service model for energy-delay optimization in edge artificial intelligence*, *IEEE Transactions on Industrial Informatics* **15** (2019), no. 7, 4216–4224.

Appendix A

The Bitcoin Simulator

We designed and developed our Bitcoin Simulator to be able to simulate Bitcoin network and Bitcoin mining process under arbitrary circumstances and analyze the results. We used this simulator to analyze the profitability of selfish mining, the effect of multiple selfish mining on the network and each other, the effect of alternative difficulty adjustment algorithms, etc. Here we want to introduce our simulator in detail and review its technical aspects.

Our simulator is a discrete-event simulator written in Python programming language (version 3) and uses a few general-purpose libraries like `numpy` and `matplotlib`, mostly for analyzing and illustrating purposes.

To be able to utilize the multi-core capabilities of modern processors and run multiple instances of the simulator to run different episode and then aggregate the results, we used Python's `multiprocessing` library¹. This way we can create multiple processes, each of which can be run on a different CPU core and simulate multiple episodes and analyze the episode. At the end, we aggregate the results for all of them to get the final results for a certain configuration.

¹Not to be confused with multi-threading. Multi-threading runs all its threads on a single process, thus all of them will run on the same CPU core, whereas multiprocessing uses different processes that can be assigned to different cores for real parallelization and better performance.

A.1 The simulator components

A simplified class diagram of the simulator is illustrated in Figure A.1. As shown in this figure, the simulator consists of several components or classes. We will talk about each of these classes in more details.

A.1.1 The Block Class

This class represents a block in the blockchain. Similar to a real block in the blockchain, it contains its `height` in the blockchain, the `difficulty` value of the network, the `time` it has been created, the `miner` who created it, and finally its parent (previous) block as `prev`. This class is a container (data type) class and does not have any notable functions.

```
1 class Block:
2     """Represents a block in the blockchain."""
3     def __init__(self, height=0, difficulty=1, time=0, miner=None, prev=None):
4         self.height = height
5         self.difficulty = difficulty
6         self.time = time
7         self.miner = miner
8         self.prev = prev
```

Listing A.1: A glimpse of the `Block` class code

A.1.2 The Event Class

As mentioned before, this simulator is a discrete-event simulator. This type of simulation, models a system as a discrete sequence of events. Each event occurs at a point in time and make changes to the state of the system. It also could create further event in the future. In this type of simulation, no changes take place in between two events. So, the simulator could jump from one event to the next event in time.

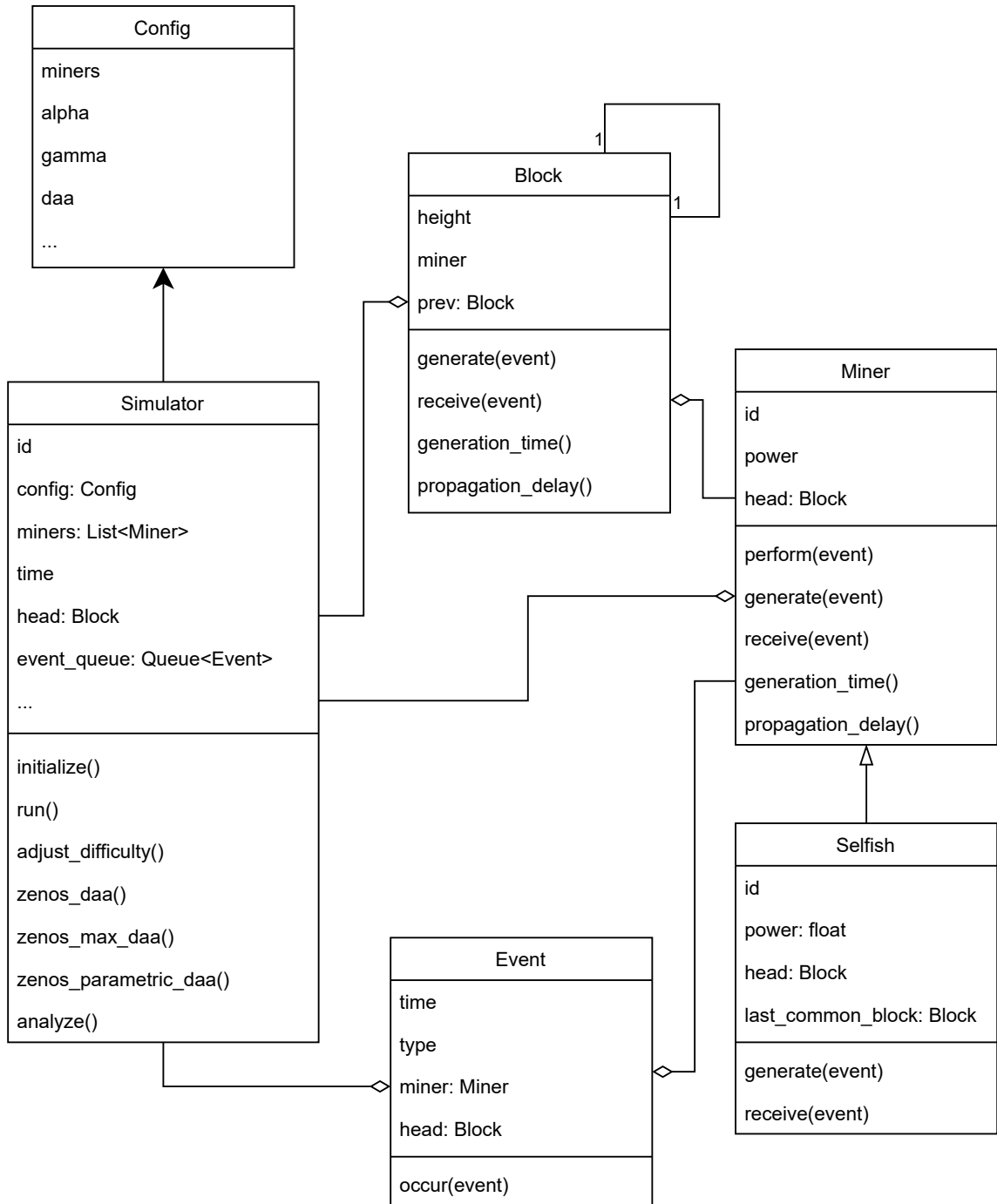


Figure A.1: Simplified Class Diagram of the Simulator

We have two types of events in the system: generate events and receive event. Generate event is when a miner provides a Proof-of-Work which means it creates a block on top of the latest block that it had. Receive event is when a miner receives a block that is generated by another miner.

As shown in Listing A.2, each event has a `time`, a `type`, a `miner` who generates or receives the block, and a `head` which is the block that has just been generated and will be the head of the blockchain. In addition, every event has a function called `occur` which is responsible for processing the event. This function simply calls the function `perform` of its miner to process the event based on its strategy and also the type of event.

```
1 class EventType(Enum):
2     NotDefined = 0
3     NewBlockFound = 1
4     NewBlockReceived = 2
5
6
7 class Event:
8     """Represents an events (generate/receive) in Bitcoin's networks."""
9     def __init__(self, time=0, type_=EventType.NotDefined, miner=None, head=None):
10        self.time = time
11        self._type = type_
12        self.miner = miner
13        self.head = head
14
15    @property
16    def is_generate(self):
17        return self._type == EventType.NewBlockFound
18
19    @property
20    def is_receive(self):
21        return self._type == EventType.NewBlockReceived
```

```

22
23     def occur(self):
24         return self.miner.perform(self)

```

Listing A.2: A glimpse of the `Event` class code

A.1.3 The Miner Class

This class models a miner in the network. Every miner has an `id`, a `power` (or hashrate), and a `head` which is the last block in the blockchain that the miner is working on top of it.

As mentioned before, there is a `perform` function in this class that is called by `occur` function in the `event` class. Based on the type of the event, this function calls one of two other functions: `generate` or `receive`. These two functions, as their names suggest, process the corresponding events. There are two other functions in this class. The first one is `generation_time` which generate a random generation time for the next block. The second one is `propagation_delay` which generates a random time for propagation of the block between miners.

```

1 import numpy
2
3
4 class Miner:
5     """Represents a miner in Bitcoin's network."""
6     def __init__(self, id_, power, head):
7         self.id_ = id_
8         self.power = power
9         self.head = head
10
11     def perform(self, event):
12         if event.is_generate:
13             if event.head != self.head:

```

```

14         # Miner is no longer mining on top of this block.
15         return [], None
16         return self.generate(event)
17     elif event.is_receive:
18         return self.receive(event)
19
20     def generate(self, event):
21         ...
22
23     def receive(self, event):
24         ...
25
26     def generation_time(self):
27         return numpy.random.exponential(600 / self.power * simulator.difficulty)
28
29     def propagation_delay(self):
30         return numpy.random.gamma(1.26, 10)

```

Listing A.3: A glimpse of the `Miner` class code

The Selfish Miner Class This class inherits from the `Miner` class. As discussed before, selfish miners diverge from the Bitcoin mining policy and adapt another strategy to increase their revenue. So, they behave differently from honest miners in the case of generating a block or receiving a block. In this class, we implemented this behaviour by overriding `generate` and `receive` functions from its parent class, `Miner`. Also, a selfish miner has a few more attributes. For example, it stores the `public` head of the blockchain (last block in the public blockchain that it has access to), the `last_common_block` between its private branch and the public branch, maximum height of the blockchain in the simulation `MAX_HEIGHT`, etc.

```

1 class Selfish(Miner):
2     """Selfish Miner!"""

```

```

3     def __init__(self, id_, power, head, simulator, **kwargs):
4         super().__init__(id_, power, head, simulator, **kwargs)
5         self.public = head
6         self.last_common_block = head
7         ...
8
9     def generate(self, event):
10        ...
11
12    def receive(self, event):
13        ...

```

Listing A.4: A glimpse of the `Selfish` class code

A.1.4 The `Config` Class

The `Config` class is responsible for storing configuration variables for the simulator. These variables define the scenario that we want to simulate using the `Simulator` class, for instance, number of miners and selfish miners and their hashrate, the type of difficulty adjustment algorithms, etc. Then an object of this class will be passed to the `Simulator` class so that it will perform the simulation.

A.1.5 The `Simulator` Class

This class is where most of the magic happens. For each episode of a scenario, we will create an instance of the `Simulator` object and feed it its configuration object. Then, we run this instance by calling its `run` function. This function starts simulating the mining process with given parameters, and continue until it reaches the desired height (or other predefined exit conditions). Then, we analyze this episode by calling the `analyze` function. This function gives us a `result` object that can be aggregated by the result of other episodes of this scenario. So, by running multiple scenarios

(for example one with a selfish miner and one without a selfish miner) for multiple episodes, we can compare the aggregated results and illustrate them.

```
1 class Simulator:
2     """The Simulator itself."""
3     instance_count = 0
4
5     def __init__(self, config=Config(), verbose=False):
6         Simulator.instance_count += 1
7         self.id = f"{Simulator.instance_count:03d}"
8         self.verbose = verbose
9         self.config = config
10        self.miners = []
11        self.time = 0
12        self.difficulty_value = 1
13        self.head = None
14        self.event_queue = None
15
16    def initialize(self):
17        ...
18
19    def run(self):
20        self.initialize()
21        breakeven = False
22        while self.head.height < self.config.MAX_HEIGHT:
23            next_event = self.event_queue.get()
24            self.time = next_event.time
25            consequences, new_head = next_event.occure()
26            ...
27            # Adapt the new head if necessary
28            # Adjust the difficulty if needed
29            # etc.
30        for c in consequences:
```

```

31         self.event_queue.put(c)
32
33     def adjust_difficulty(self):
34         ...
35
36     def zeno_daa(self):
37         ...
38
39     def zeno_max_daa(self):
40         ...
41
42     def zeno_parametric_daa(self, tau=.5):
43         ...

```

Listing A.5: A glimpse of the `Simulator` class code

A.1.6 Other components

There are several other components in the project that are not shown in Figure A.1 for the purpose of simplifying this section. For example there is a component that is responsible for multi-processing, assigning simulators to different processes and gathering the results. Also there are two other components, one responsible for aggregating the results of simulation episodes and the other one illustrating the results in different plots and figures. These components are also very important in performance and usability of our simulator. However, as they are not related to core functionality of the simulator, we skip discussing them further to keep this appendix brief and relevant.

Vita

Candidate's full name: Hamid Azimy

University attended:

University of New Brunswick, Fredericton, NB, Canada
PhD Candidate, Faculty of Computer Science, started May 2017
Dissertation Topic: Difficulty Adjustment Algorithms for Preventing Proof-of-Work Mining Attacks
Supervisors: Dr. Ali A. Ghorbani, Dr. Ebrahim Bagheri

University of Tehran, Tehran, Iran
Master of Science, School of Electrical and Computer Engineering, 2015
Supervisor: Dr. Masoud Asadpour

Iran University of Science and Technology, Tehran, Iran
Bachelor of Science, School of Computer Engineering, 2012
Supervisor: Dr. Behrouz Minaei

Journal Publications:

Hamid Azimy, Ali Ghorbani, and Ebrahim Bagheri. “*Preventing Proof-of-Work Mining Attacks*”. Information Sciences 608 (2022): 1503-1523.

Conference Publications:

Hamid Azimy and Ali Ghorbani. “*Competitive selfish mining*.” In 2019 17th International Conference on Privacy, Security and Trust (PST), pp. 1-8. IEEE, (2019).

Hamid Azimy and Ali Ghorbani. “*Alternative Difficulty Adjustment Algorithms for Preventing Selfish Mining Attack*.” In International Conference on Blockchain, pp. 59-73. Springer, Cham, (2021).