Information Sciences 608 (2022) 1503–1523

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Preventing proof-of-work mining attacks

Hamid Azimy [a,*], Ali A. Ghorbani [a], Ebrahim Bagheri [b]

[a] Canadian Institute for Cybersecurity, Faculty of Computer Science, University of New Brunswick, Fredericton, NB, Canada
[b] Department of Electrical, Computer, and Biomedical Engineering, Ryerson University, Toronto, ON, Canada

ARTICLE INFO

Article history:
Received 6 May 2021
Received in revised form 30 June 2022
Accepted 5 July 2022
Available online 9 July 2022

Keywords:
Bitcoin network
Blockchain
Selfish mining
Difficulty adjustment algorithm

ABSTRACT

Bitcoin mining is the process of generating new blocks in the Bitcoin blockchain. This process is vulnerable to different types of attacks. One of the most famous attacks in this category is *selfish mining*. This attack is essentially a strategy that a sufficiently powerful mining pool can follow to obtain more revenue than its fair share. The reason that selfish mining is effective is the difficulty adjustment algorithm used in the Bitcoin network. In this paper, we analyze the profitability of selfish mining with respect to time and propose an alternative difficulty adjustment algorithm that discourages selfish mining while allowing the Bitcoin network to remain scalable. We analyze our proposed solution, present the results, and discuss its effectiveness. Based on our analysis, our proposed algorithm effectively increases 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, the algorithm extends the waiting time from 4 weeks to more than 11 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.

1. Introduction

One of the goals of the Bitcoin network is to keep the *block generation rate* constant. This means that the whole network should be able to generate a block every ten minutes on average. So, if the network grows, difficulty needs to also increase proportionally and vice versa. In practice, this happens every 2016 block, which is roughly equivalent to two weeks. This process is known as the *difficulty adjustment* process.

Bitcoin's Difficulty Adjustment Algorithm (DAA) is very simple, i.e., at the end of every 2016 block period, all the nodes in the Bitcoin network calculate the ratio of *expected block generation rate* to the *actual block generation rate*. This ratio offers the actual difficulty value for the previous period, which is then multiplied by the current difficulty value to produce the next difficulty value. By doing so, the *difficulty value* of the next period is set proportional to the actual *difficulty value* of the previous period.

Given the simplicity of the difficulty adjustment algorithm, some malicious Bitcoin network users have exhibited behaviour that is often known as *selfish mining*, first introduced by Eyal and Sirer [12]. Selfish mining attempts to maximize the profitability of the attacker through strategies such as hiding generated blocks from the main blockchain. There has been conflicting evidence on how effective selfish mining attacks could be. As our **first contribution**, we investigate the effective-

* Corresponding author.
E-mail addresses: hazimy@unb.ca (H. Azimy), ghorbani@unb.ca (A.A. Ghorbani), bagheri@ryerson.ca (E. Bagheri).

https://doi.org/10.1016/j.ins.2022.07.035
0020-0255/

ness of selfish mining and show that *time* is an essential factor in the profitability of selfish mining, and attackers who are powerful enough can potentially benefit from selfish mining after a certain waiting period.

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 **second contribution**, we propose an alternative difficulty adjustment method 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 proposed 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.

As our **third contribution**, we developed a Bitcoin network simulator. This simulator, which will be discussed in more detail in Section 6.3, 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 employed 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, for our **fourth contribution**, we used this simulator to study and analyze selfish mining and the effectiveness of our proposed DAA.

To summarize, here is the list of contributions of this paper:

1. An empirical study to show the effect of *time* on the profitability of selfish mining;
2. Proposing an alternative DAA to overcome the issues with the default DAA and discouraging selfish mining;
3. Developing a Bitcoin network simulator to simulate different mining strategiesand difficulty adjustment algorithms;
4. Conducting experiments and analyzing the results to show the effect of time on selfish mining and the effectiveness of the proposed DAA.

The rest of the paper is organized as follows. In Section 2, we will go through the related works, review threats and attacks to the Bitcoin network and blockchain, and discuss some countermeasures. In Section 3, we will analyze selfish mining and discuss its profitability. The problem will be defined in Section 4, and our proposed method will be introduced in Section 5. In Section 6, we will evaluate our proposed method and present the results. We also discuss the properties of our proposed method in Section 7. Finally, in Section 8, we will conclude our work and discuss potential future research.

2. Literature Review

In recent years, blockchain has received growing attention [4] in domains such as financial services, insurance, healthcare, voting systems, IoT, and smart city, could storage [21], to name a few. This naturally leads to a variety of threats towards the blockchain structure and its mining process [38]. In a broader view, we can categorize Bitcoin threats into two major categories: *privacy issues* and *security issues*. Conti et al. [7] have surveyed these issues in their paper. We will selectively review these issues as well as a few example attack types in the next sections.

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 users' privacy. 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; 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 Bitcoin users are associated with Bitcoin addresses, which are linked to a pair of Elliptic Curve Digital Signature Algorithm (ECDSA) public and private keys. Anyone can generate any number of Bitcoin addresses and use them. Thus every node in the network can see that someone (with a public address) is sending some bitcoins to someone else (also with a public address), but these addresses cannot be easily linked to a real identity. Also, to further improve privacy, it is strongly suggested that users should use each address only once. However, this is an ideal scenario; In the real world, linking public addresses to individuals is possible to some extent. Adversaries can perform blockchain analysis to follow the flow of the coins in the Bitcoin network using some simple rules. For example, in multi-input transactions, all input addresses almost certainly belong to a single individual [26], or the address provided to collect the change (after transferring the specified amount to the receiver) also belongs 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 [23]. Androulaki et al. performed an experiment to deanonymize users in such networks [1], and showed promising results.

Therefore it is often stated that Bitcoin does not offer anonymity. Instead, it offers *pseudonymity*. This problem is not limited to Bitcoin. Many other blockchains and cryptocurrency systems suffer from the same problem [19]. There have been

various efforts to mitigate privacy issues in Bitcoin [22,18]. They can be categorized into three major categories: *Peer-to-peer mixing protocols*, *Distributed mixing networks*, and *Altcoins* [7].

- **Peer-to-peer mixing protocols:** Mixing is a technique that can be used to confuse the trail of transactions to remove linkability in the Bitcoin network 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, and 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 users' privacy. There are many examples of these mixing protocols, e.g., CoinJoin [25] and CoinShuffle [36] among others.
- **Distributed mixing networks:** In this approach, the users share some bitcoins through a third party and receive back the same amount of coins. So it provides strong anonymity. MixCoin [5] is one of the examples in this category. 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 [42] uses blind signatures to create user inputs and cryptographically blinded outputs called blinded tokens. Another example is TumbleBit [14], which helps users to make fast, off-blockchain payments anonymously through an intermediary called Tumbler.
- **Altcoins:** While Bitcoin remains the first and most widely used cryptocurrency in the world, there are hundreds of other coins that are inspired by Bitcoin. These coins are called *altcoins*. Some of these altcoins try to tackle privacy issues and offer different mechanisms to ensure the anonymity of their users. One of these altcoins is ZeroCoin [28] that uses cryptographic techniques such as zero-knowledge proof to validate encrypted transactions. Two extensions of ZeroCoin are ZeroCash [40] and Zcash [17], 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* which mixes transactions using master nodes to achieve higher privacy.

2.2. Security of Bitcoin

Our focus in this paper 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. Therefore, first, we discuss pooled mining and different types of attacks in this category.

2.2.1. Pooled mining

As the size of the Bitcoin network grows, its computational power also increasesTherefore,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 extremely limited in comparison to the computational power of the network. Thus, an individual has a small chance of finding a block. However, 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 [50,49].

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 very rare occasions, she will receive a significant reward. To reduce income fluctuations, 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 significantly reduce the fluctuation of the reward, 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.

2.2.2. Mining attacks

There are several categories of attacks against the Bitcoin network and other blockchains in general [8]. 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 transactions and Bitcoin network, meaning they are not trying to compromise the validity of the blockchain or steal bitcoins. Instead, they target the miners and pools. In a scenario where all miners are working honestly (loyal to the Bitcoin protocol), in the long run, any miner will receive a reward proportional to her computational power[1]. For example, suppose that the total computational power of the entire Bitcoin network is one. Then suppose one miner or a pool of miners have 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.

One of the most famous attacks in this category is '*Selfish Mining*' introduced by Eyal and Sirer [11]. In selfish mining, the attacker intentionally creates forks. It means 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. We will discuss this attack in more detail in the following sections. The importance of selfish mining is that it is not limited to Bitcoin, and it can be applied in other blockchains,

[1] Terms '*computational power ratio*' and '*hashrate*' both refer to the ratio of the number of hashes that a miner can generate per unit of time to the total number of hashes that can be generated by the whole network. We use this two terms interchangeably throughout this paper.

e.g., Ethereum [33,24]. Based on [44], 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 [37]. However, the likelihood of such attacks is increasing with the growth in the Bitcoin network. Many other attacks in this category try to expand or generalize the selfish mining attack or combine it with other attacks [3].

Block withholding attack, introduced by Rosenfeld [35], is another 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.

Stubborn mining, presented by Nayak et al. [31] in 2015, tries to improve selfish mining by a new strategy. As the name suggests, the attacker acts more stubbornly in this strategy and keeps mining on her private branch even though it is not the longest branch. They also combine it with another network-level attack called Eclipse attack to make it even more profitable. 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. This attack was introduced by Heilman et al. [15].

The next attack is *optimal selfish mining*. In this attack, Sapirshtein et al. (2016) [39] provide an algorithm to find an optimal selfish mining policy that guarantees a lower bound for the revenue. Another attack in this category is Fork After Withholding attack, introduced by Kwon et al. [20] in 2017. This attack combines Selfish Mining and Block Withholding attacks to overcome their limitations and create a stronger attack.

Meshkov et al. [27] introduced a new attack called Coin–hopping. In this attack, an attacker switches between two coins, C_1 and C_2. Suppose that at first, the attacker has not joined any of the coins yet, and the difficulty of both coins is *one*. If the attacker joins one of the coins, the difficulty of that coin will increase because of the network growth. Now, the attacker will join C_1 for one difficulty adjustment period, and therefore mine C_1 with the difficulty value of *one*. Because of this, for the next period, the difficulty of C_1 will increase to a value greater than *one*; but the difficulty of C_2 remains at *one*. So for the start of the second period, the attacker switches to C_2 with the difficulty of *one* for one period. Because of this switch, now the difficulty of C_2 will increase to a value greater than *one*, but meanwhile, the difficulty of C_1 will go back to *one*, because the attacker left the C_1's network. So the attacker comes back to C_1, and so on. Therefore, the attacker always mines on a coin with the difficulty of *one*, while if she stayed at one coin, she should have mined on a coin with difficulty greater than one, which is not as profitable for her.

2.2.3. Selfish mining

As discussed earlier, Selfish Mining [12], is an attack from a mining pool toward other miners or mining pools. Selfish miners do not follow the protocol. Instead, they follow a set of rules that help them to 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 keep it for herself, so other miners cannot mine on the top of her block, which is the last block in the blockchain. By doing so, she creates a private branch alongside the public branch (that is accessible to everyone). Then she publishes her private branch whenever it's needed according to a specific set of rules to invalidate honest miners' blocks. This set of rules is called selfish strategy and it indicates which action should be taken by the selfish miner based on the state of the network (length of the branches) and the last event in the network (either the new block is mined by the selfish miner or other miners). By following the selfish strategy, the attacker will waste a lot of computational power from honest miners. There is multiple research that analyzes selfish mining in different circumstances [47,29]. For example, Motlagh et al. analyzed the effect of selfish mining on the performance of the network [30]. Wang et al. analyzed the impact of propagation delay and network distance on the performance of selfish mining [45] Also, there are several researchers that have analyzed selfish mining in a network with multiple selfish miners [2,48].

Because of the decentralized nature and anonymity (pseudonymity) of Bitcoin and blockchain, detecting selfish mining is not so straightforward. However, there have been several attempts to detect this attack [6]. For instance, Saad et al. [37] leveraged the expected transaction confirmation height and the block publishing height to detect selfish mining behaviour in Proof of Work (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 a few mitigation techniques proposed to prevent mining attacks [34]. Our focus here is on selfish mining prevention. The first mitigation strategy, proposed by Eyal and Sirer [12], suggests a slight change in the Bitcoin protocol. Bitcoin protocol requires that when receiving two blocks with the same height, a node should accept the first one, broadcast it to others, and ignore the second block completely. In this mitigation technique, Eyal and Sirer suggest that in such situation, the node should broadcast both of those blocks, and choose one of them *randomly* to mine on. They prove that by using this technique, we can 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, 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 [14]. In this method, every block should have a timestamp. These timestamps are unforgeable (meaning no one can 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. 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, and it (as the method name suggests) prefers the fresher one. Because if a selfish pool withholds a block that it had 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 [41]. 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 listen 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 min, which is average block generation time in Bitcoin), she generates a Zeroblock (which does not need a PoW, and is easy to generate, and can be created by all nodes independently), and appends it to its local version of the blockchain. From now on, this block is the new head of the blockchain, and all miners should mine on top of this block, instead of the previous block. If she receives a withheld block in the future, she will know that this is withheld because it is not mined on the top of the Zeroblock. 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 devaluate the whole network.

There are a few problems with these mitigation techniques. More specifically, in freshness preferred, we need an unforgeable timestamp, and therefore, an entity that generates these timestamps. First, having such an entity challenges 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 adding a dummy block to the blockchain, because this dummy block cannot include any transactions, we have periods that no transactions can be confirmed. Thus the transaction confirmation time for those transactions will increase by at least the time of the zeroblock. Another problem with these mitigation techniques is that they are fairly complicated and need significant changes 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.

2.3. Blockchain Trilemma

Researchers have already discussed that the blockchain or essentially any distributed system can face a trilemma often known as the *scalability trilemma* [43,16]. 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 process higher transaction throughput. This component 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.

3. Profitability of Selfish Mining

In this section, we will review countering arguments on the profitability of selfish mining. As one of the earlier works on this topic, Eyal and Sirer analyzed selfish mining and presented it as a state machine with the transition probabilities between the different states [12] Fig. 1. Using this state machine, they computed the state probabilities [12] Eq. (2)–(5). In their analysis, they use two parameters. The first parameter is α which is the selfish pool's hashrate, relative to the hashrate of the whole network. Consequently, the hashrate of all the other miners combined is $1 - \alpha$. The next parameter is γ, which denotes the ratio of honest miners who choose to mine over selfish mining branch. 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, its revenue and also the revenue of honest miners should be calculated. The revenue of a miner is the sum of products of state probabilities and state revenues, over all the states. Based on different cases that might happen in every state, Eyal and Sirer calculated the revenues [12] Eq. 6,7 and called them r_{pool} and r_{others}, which show the revenues of the selfish pool and other (honest) miners, respectively.

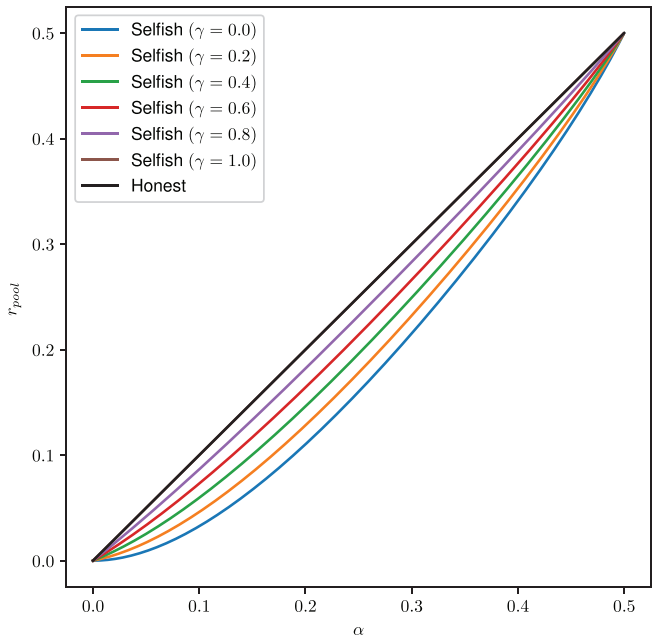


Fig. 1. r_{pool} for different α and γ (after difficulty adjustment).

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 as '*effective hashrate*' (or '*apparent hashrate*' [13]) of the pool in this scenario.

Fig. 1 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 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. Therefore from the perspective of the total net revenue, selfish mining is never profitable.

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

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.

$$r_{total} = r_{pool} + r_{others} < 1$$

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 chose to use this strategy 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 compared to Bob's, and now she makes twice as Bob's. But her total net revenue decreased by one bitcoin per day, which is not desirable for a rational agent. Also, the total revenue in this example is reduced from ten to six.

Wright and Savanah [46] argue that this scenario will always hold. 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. However, in practice, the Bitcoin network operates based on 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 by the sum of the revenues of both honest miners 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. This results in:

$$R_{pool} = \frac{r_{pool}}{r_{pool} + r_{others}} = \frac{r_{pool}}{r_{total}} \tag{1}$$

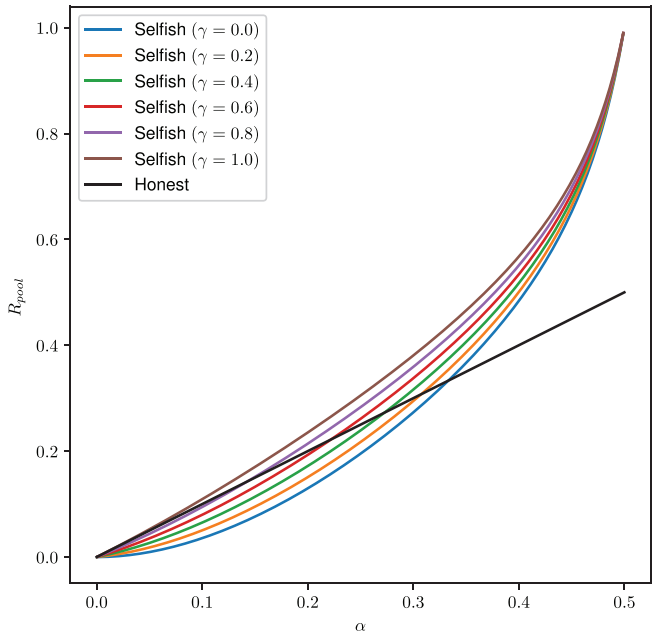


Fig. 2. R_{pool} for different α and γ (after difficulty adjustment).

Fig. 2 shows the value of R_{pool} with the same configuration as Fig. 1. It shows that in many cases, the pool's revenue exceeds its revenue in case of honest mining. This is a strong motivation to perform this attack against other miners in the network. The fact that **difficulty adjustment makes selfish mining profitable** is a crucial point in this paper, and we will get back to it in the following sections.

An important factor that neither Eyal and Sirer [12] nor Wright and Savanah [46] considered in their analyses is *time*. Eyal and Sirer considered the final value of the revenue, while Wright and Savanah considered just the first period before any difficulty adjustments. 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 above, regardless of the selfish miner's hash power, selfish mining is not profitable at the beginning. However, after the difficulty adjustment, the revenue of the selfish miner starts to increase. If the selfish miner is powerful enough, it will eventually exceed the revenue of honest mining strategy. We will analyze this in detail in Section 5.

4. Problem statement

As we discussed in the previous section, an important factor that makes selfish mining profitable is the *difficulty adjustment* algorithm. That is because instead of increasing the pool's revenue, selfish mining 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 will become profitable in terms of *total net revenue*.

As discussed in Section 1, difficulty adjustment is a crucial mechanism to ensure the scalability of the Bitcoin network. Without the 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 should be difficult 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, in Section 3, we discussed that 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. 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.

One of the contributions of this paper is a solution to this problem. We need an alternative difficulty adjustment algorithm that can maintain the scalability of the network while discouraging dishonest mining behaviour.

5. Proposed approach

The first step toward our goal, which is to propose an alternative difficulty adjustment algorithm, is to understand the features and properties of a good difficulty adjustment algorithm. For example, similar to the original Bitcoin difficulty

adjustment algorithm, it should be easy to understand, implement, and compute. Everyone should be able to compute it by themselves without any need to interact with others, just by having the blockchain history to this point. For this step, we will discuss a list of properties of a sound difficulty adjustment mechanism.

For example, Meshkov et al. describe the properties of an ideal difficulty adjustment algorithm as follows [27]:

1. It should make Bitcoin mining resistant to the attacks that work based on manipulating difficulty.
2. It should make the block generation rate constant and as close to the desired value as possible.

However, the attack that Meshkov et al. had in mind was the coin–hopping attack, which is different from the selfish mining attack considered in this paper. In the coin–hopping attack, the attacker's focus is to alternate the difficulty between a higher value and a lower value. As such, she will mine on lower difficulty periods and switches the network on higher difficulty periods. Therefore, a linear DAA proposed Meshkov et al. is able to address such an attack.

On the other hand, 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 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 attacks. 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 allowing miners to perform selfish mining attacks 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 adding a very complicated module to the Bitcoin system just for the difficulty adjustment. It should also 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.

To summarize, based on the above discussion, a sound DAA for our purpose should:

(P_1) be **resistant to selfish mining** and ideally, other types of attacks based on difficulty manipulation.
(P_2) lead to an **almost constant block generation rate** as close to the desired value as possible.
(P_3) be able to **maintain the scalability** of the network.
(P_4) be **easy to understand, implement, and calculate**.
(P_5) be **independently computable by peers**.

To be able to analyze difficulty adjustment algorithms, 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 miners' *relative revenue* will increase. But that is not desired, because a miner does not want her total net revenue decrease, even if her relative revenue increases.

Since the number of bitcoins mined is directly related to the number of blocks generated (currently 6.25 bitcoins per block), we simply 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.

$$R_{P_i}(t): \begin{array}{l}\text{Number of blocks generated by} \\ \text{the pool } P_i \text{ in time } t \\ (\text{Total net revenue of the pool } P_i)\end{array} \tag{2}$$

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 $\overline{R}$ as below.

$$\overline{R}_{P_i}(t) = \alpha_i \times t \times 6 \tag{3}$$

The constant value of 6 in the equation above indicates that, on average, six blocks should be generated by the whole network every hour. In other words, the block generation rate for Bitcoin is ten minutes.

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

$$R_{P_i}(t) \approx \overline{R}_{P_i}(t), \text{if all miners are honest} \tag{4}$$

By comparing R_{P_i} and $\overline{R}_{P_i}$ 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 '*gain*' for short, denote it by G_{P_i} and defined as follows:

$$G_{P_i}(t) = \frac{R_{P_i}(t) - \overline{R}_{P_i}(t)}{t \times 6} \tag{5}$$

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

We further assume that 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 block, and the block generation rate is one every ten minutes or $\frac{1}{6}$ hours. So the length of a difficulty adjustment period should be as follow:

$$T = 2016 \times \frac{1}{6} = 336 \; hours \tag{6}$$

Suppose that at the beginning, the value of difficulty equals one. When a pool starts to perform the selfish mining strategy, the value of the difficulty stays equal to one for the first period. Meanwhile, selfish mining creates a lot of discarded blocks, which leads to a 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}.

$$T_1 = \frac{T}{r_{total}} > T \tag{7}$$

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.

$$\begin{aligned} &D_0 = 1 \text{ (initial difficulty)} \\ &D_1 = D_0 \times r_{total} \\ &D_{k>1} = D_1 \end{aligned}$$

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

$$D(t) = \begin{cases} 1, & t < T_1 \\ r_{total}, & t \geqslant T_1 \end{cases} \tag{8}$$

Fig. 3 shows the value of difficulty with respect to time.

Now, we calculate Eq. 1, but this time we consider the time and this means we have to consider the effect of the difficulty adjustment:

$$\begin{aligned} R_{pool}(t) &= \int_0^t \left(\frac{r_{pool}}{D(t)} \times 6\right) dt \\ &\text{if } t > T_1 \Rightarrow \\ R_{pool}(t) &= \int_0^{T_1} \left(\frac{r_{pool}}{1} \times 6\right) dt + \int_{T_1}^t \left(\frac{r_{pool}}{r_{total}} \times 6\right) dt \\ &= \left[\frac{r_{pool}}{1} \times 6t\right]_0^{T_1} + \left[\frac{r_{pool}}{r_{total}} \times 6t\right]_{T_1}^t \\ &= r_{pool} \times 6T_1 + \frac{r_{pool}}{r_{total}} \times 6(t - T_1) \\ &= \frac{r_{pool}}{r_{total}} \times 6t - \left(\frac{r_{pool}}{r_{total}} - r_{pool}\right) \times 6T_1 \\ &= R_{pool} \times 6t - \left(R_{pool} - r_{pool}\right) \times 6T_1 \end{aligned} \tag{9}$$

In Eq. 9, as seen previously, r_{pool} and R_{pool} show the effective hash rates of the pool before and after the difficulty adjustment, respectively. Also, from Eq. 9, we can conclude that if $t \longrightarrow \infty$, the value of $R_{pool}(t)$ would converge to R_{pool} from Eq. 1, 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.

We also have:

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

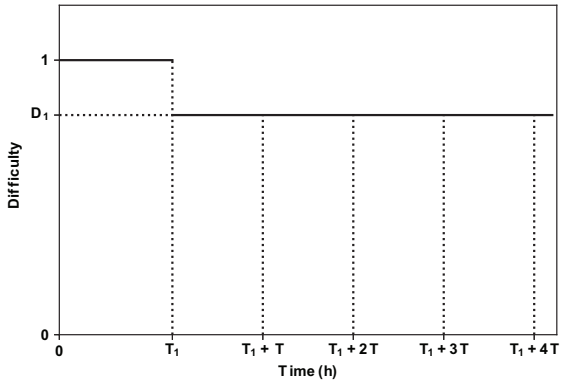


Fig. 3. Difficulty value by time in a selfish mining scenario.

Therefore:

$$\begin{aligned} G_{pool}(t) &= \frac{R_{pool}(t)-\bar{R}_{pool}(t)}{6t} \\ &= \frac{R_{pool}(t)-\alpha\times 6t}{6t} \\ &\text{if } t > T_1 \Rightarrow \\ G_{pool}(t) &= \frac{\left(R_{pool}\times 6t-\left(R_{pool}-r_{pool}\right)\times 6T_1\right)-\alpha\times 6t}{6t} \\ &= \left(R_{pool}-\alpha\right)+\left(R_{pool}-r_{pool}\right)\times\frac{T_1}{t} \end{aligned} \tag{10}$$

Fig. 4 shows an example of $G_{pool}(t)$ for a selfish miner with hashrate of 0.35 (or %35). As shown 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 [46] 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 twelve weeks from the beginning of the attack), the selfish strategy evens out. We call this point a *break-even point*. After that, it continues increasing slowly and eventually, it converges to the point that Eyal and Sirer predicted in [12] Eq. 8. This figure clearly shows that *time* plays an important role in the profitability of selfish mining.

Based on the properties of a sound DAA, we propose a novel difficulty adjustment algorithm and analyze its properties and performance. The formula of Bitcoin's current DAA is as follows:

$$D_n = D_{n-1}\times\frac{2016\times 600}{t_{b_{2016\times n}}-t_{b_{2016\times(n-1)}}} \tag{11}$$

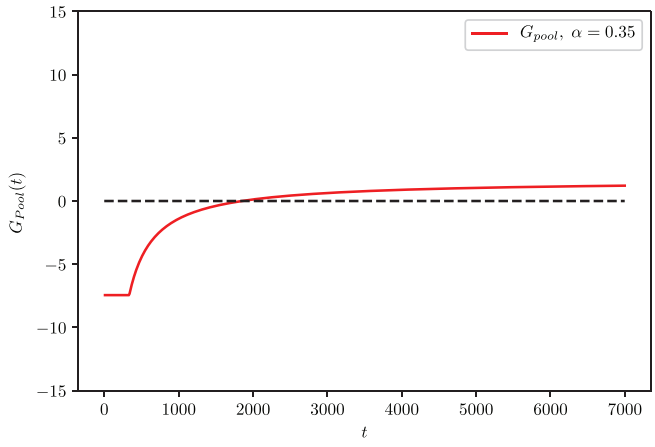


Fig. 4. Gain for a selfish miner by time.

in which, D_n is the difficulty from the n'th period forward, b_i is the i'th block, and t_{b_i} is the generation time for block i in seconds. Also, the numerator is the expected time for generating 2016 blocks for an average of 600 s (10 min).

Our proposed method modifies the original DAA of Bitcoin to make the difficulty sensitive to history, not just the expected difficulty of the current period. We formulate it as follows:

$$E_{n-1} = D_{n-1} \times \frac{2016 \times 600}{t_{b_{2016\times n}} - t_{b_{2016\times(n-1)}}} \tag{12}$$

$$D_n = \frac{1}{2} E_{n-1} + \frac{1}{2} D_{n-1}. \tag{13}$$

Here, E_{n-1} is the expected difficulty for the previous period, and D_{n-1} is the difficulty value that has been used for this period. But, unlike what the current DAA does, we do not set this value as the next period's difficulty. Instead, we set the difficulty of the next period to a simple average between these two values.

This formulation considers a history of difficulties and therefore changes more smoothly than the original one. 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. In the evaluation section, we will discuss how our proposed DAA satisfies each of the properties P_1 through P_5.

6. Evaluation and Results

In this section, we evaluate our proposed difficulty adjustment algorithm, Zeno's DAA, and discuss how it satisfies the properties we discussed in the previous section. We compare Zeno's DAA with Bitcoin's default difficulty adjustment algorithms to see its effect on selfish mining reward and block generation rate. In the literature, there have been a few works that compare existing difficulty adjustment algorithms from different proof-of-work cryptocurrencies with the highest market share (e.g. Bitcoin Cash, Litecoin, Monero, Dash, Zcash) [32,9]. Based on these studies, although other DAA's might have their advantages form other perspectives, Bitcoin's DAA is the one that shows the most resistance to selfish mining and makes selfish mining less profitable. Therefore, we compare our proposed algorithm with Bitcoin's DAA and try to outperform it.

6.1. Research Questions

In this paper, we introduced an alternative difficulty adjustment algorithm that discourages selfish behaviour among miners and mining pools. In Section 5, 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 an adjustment algorithm, referred to as Zeno's DAA, in such a way that it extends 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 paper. 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?

6.2. Evaluation Metrics

In Section 5 we talked about *total net revenue* and also *gain* as indicators to evaluate profitability of selfish mining in different circumstances. We use these two metrics here to evaluate our DAA's effect on the profitability of selfish mining, more specifically, the waiting period before it becomes profitable. We defined these two metrics in Eq. 2 and Eq. 5 and used them to evaluate property P_1. Another metric, which we use to evaluate P_2 is the block generation rate. We will use the average block generation rate with respect to time and compare it to the desired value to examine the effect of the new DAA on block generation rate.

[2] https://gitlab.com/hamidazimy/bitcoin-simulator

6.3. Simulator

We developed and used a simulator to simulate different scenarios and analyze their outcome. Our simulator[2], which is written in Python 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 studied the effect of multiple selfish miners on the Bitcoin network [2].

The simulator uses an event queue. This event queue is a priority queue, in which the priority of every event is its occurrence time. Each step of the simulation in this simulator is as follows: 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 the 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. For every event added 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 a miner receiving a new block 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 [10], in which they analyzed information propagation in the Bitcoin network. We also 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, based on the Bitcoin protocol, the miner will accept the new block (of course, if it contains valid transactions) 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 circumstances. We implemented the strategy described in Section 2.2.3 and used this new miner to analyze the selfish mining attack.

After running the simulator for the desired amount of time, we can get the simulation outcome and analyze them in order to get our desired results.

6.4. Findings

In this section, we use our simulator to analyze 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 compared the results.

6.4.1. Simulation results for selfish mining

For the case that all the miners are honest and loyal to the protocol, Fig. 5 shows the outputs of Eq. 2 and also the results of the simulation[3], which demonstrates $R_{P_i}(t)$ from Eq. 14, for different values of α. As seen in these figures, the two curves in both figures match perfectly.

$$\overline{R}_{P_i}(t) \approx \alpha_i \times t \times 6 \tag{14}$$

$$R_{P_i}(t) \approx \overline{R}_{P_i}(t), \text{if all miners are honest} \tag{15}$$

The desired value for $G_{P_i}(t)$ from Eq. 16 is *zero* when all miners are honest. Fig. 6 shows this value for honest miners with different hashrates.

$$G_{P_i}(t) = \frac{R_{P_i}(t) - \overline{R}_{P_i}(t)}{t \times 6} \tag{16}$$

The next result of the simulator is related to Eq. 17 (Eq. 6 from Section 5):

$$T = 2016 \times \frac{1}{6} = 336 \; hours \tag{17}$$

In an *all-honest scenario*, we expect the value of T to be very close to 336 h. We ran the simulator 350 times, and Fig. 7 shows the histogram of different values of T from these simulations. Also, the mean and variance of the samples are as follows:

$$\mu = E[T] = 336.22$$
$$\sigma^2 = Var(T) = 63.76$$

[3] All the simulation results here are average between 35 independent simulations.

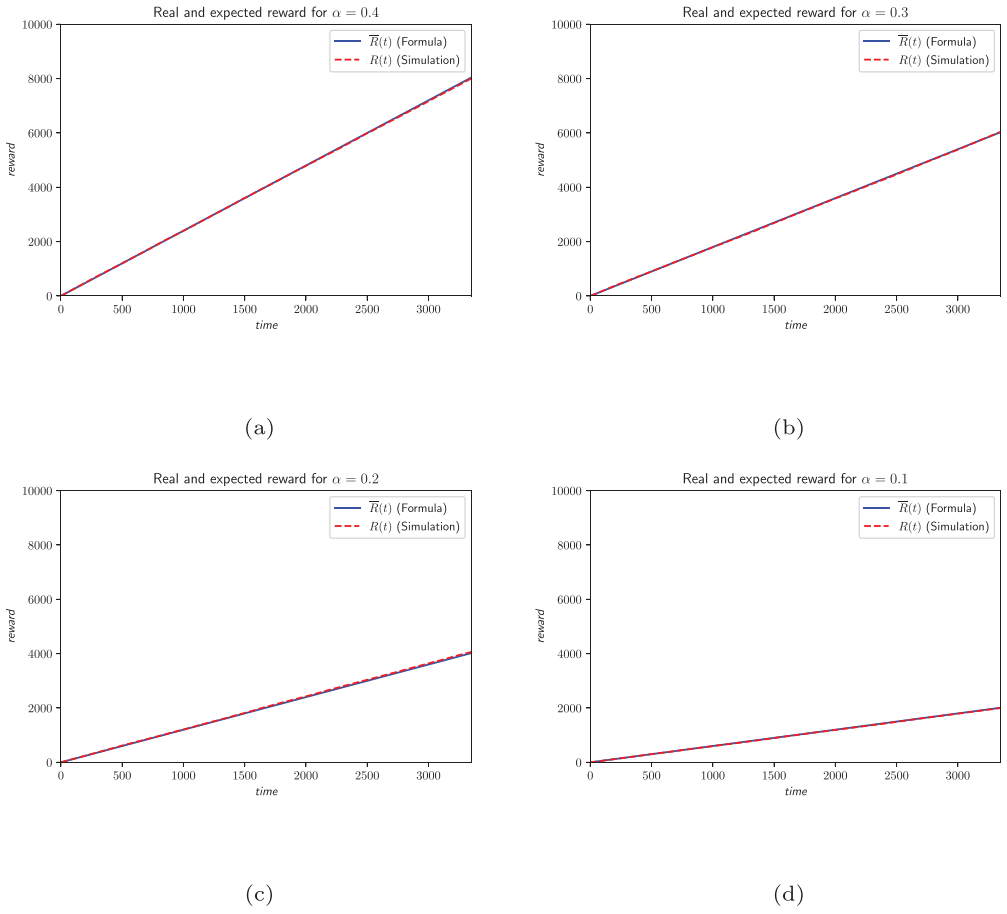


Fig. 5. Real and expected rewards for honest miners with different hashrates.

T_1 is the length of the first difficulty adjustment period in the presence of a selfish miner. Fig. 8 shows the value of T_1 for different network configurations with selfish miners who have hash rates between 0.25 and 0.50. As seen, the two curves match perfectly.

$$T_1 = \frac{T}{r_{total}} > T \tag{18}$$

In Eq. 19, we have the value of *the difficulty*, in the case that the selfish miner starts at time *zero* with no other changes in the network.

$$D(t) = \begin{cases} 1, & t < T_1 \\ r_{total}, & t \geqslant T_1 \end{cases} \tag{19}$$

Fig. 9a,9b show $D(t)$ for two different network configurations with $\alpha = 0.30$ and $\alpha = 0.45$, respectively. Again in these figures, 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 a direct relationship with α and the final value of D has an inverse relationship with it).

The next equation is Eq. 20, which is defined as follows:

$$R_{pool}(t) = R_{pool} \times 6t - (R_{pool} - r_{pool}) \times 6T_1 \tag{20}$$

Fig. 10,11, show the values of $R(t)$ (from the simulation and the equation) and $\overline{R}(t)$ (expected value of reward in all honest scenario) for two network configurations of $\alpha = 0.40$ and $\alpha = 0.33$, respectively.

The last equation, which is the most important for us, is Eq. 21:

$$G_{pool}(t) = (R_{pool} - \alpha) + (R_{pool} - r_{pool}) \times \frac{T_1}{t} \tag{21}$$

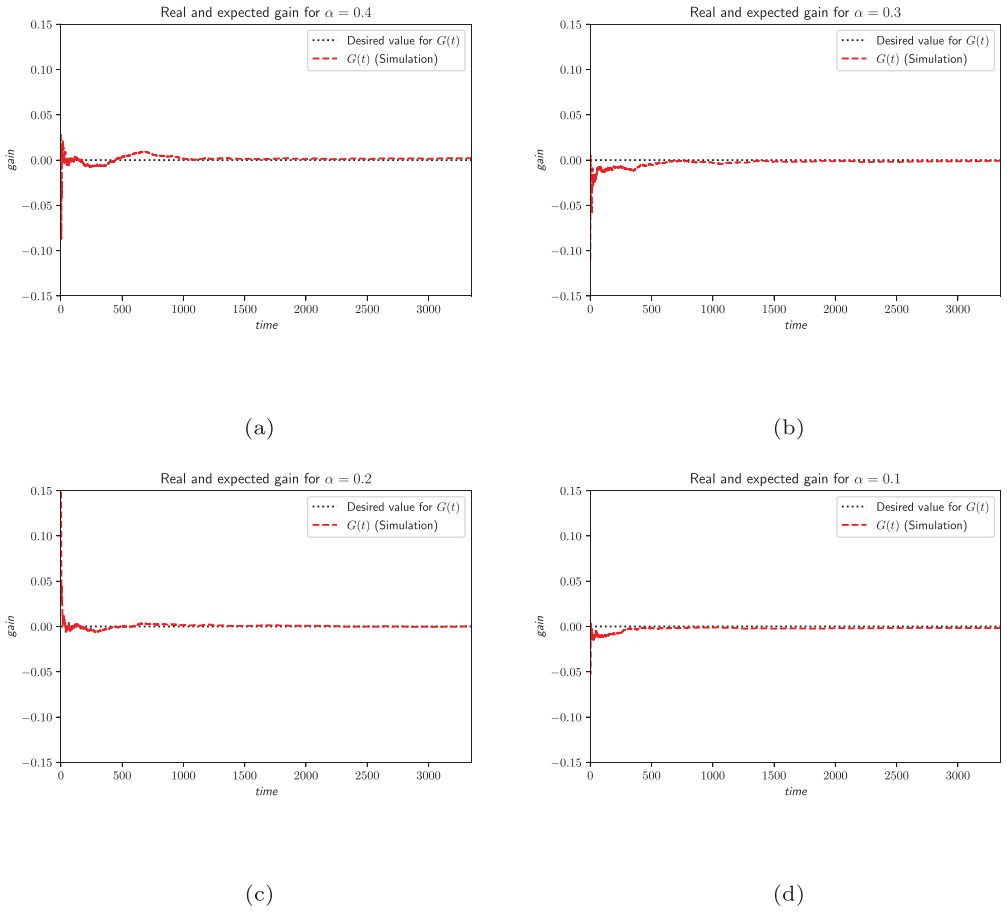


Fig. 6. Real and expected gain for honest miners with different hashrates.

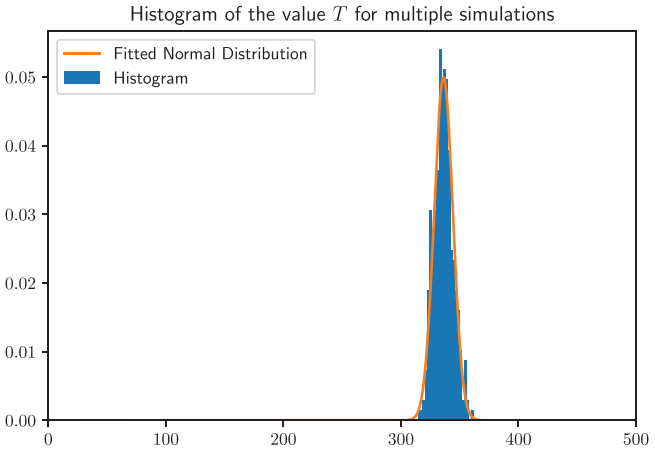


Fig. 7. Histogram of value *T*.

Similar to the previous figures, Fig. 12,13, show the values of $G(t)$ from the simulation and the equation 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 the all-honest scenario.

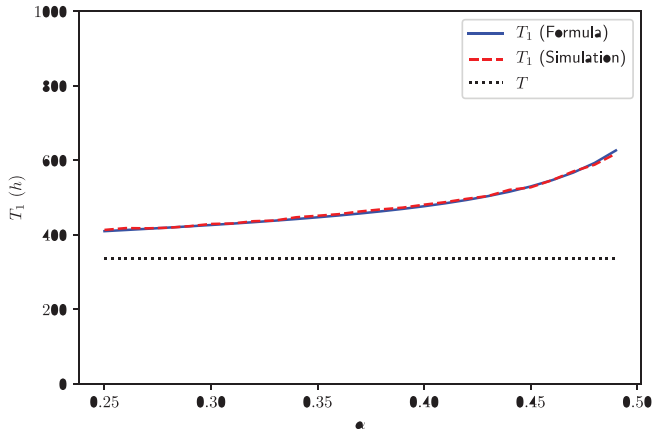
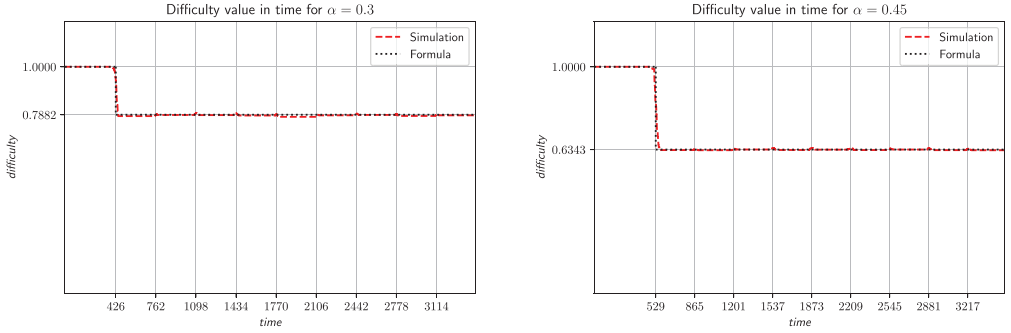
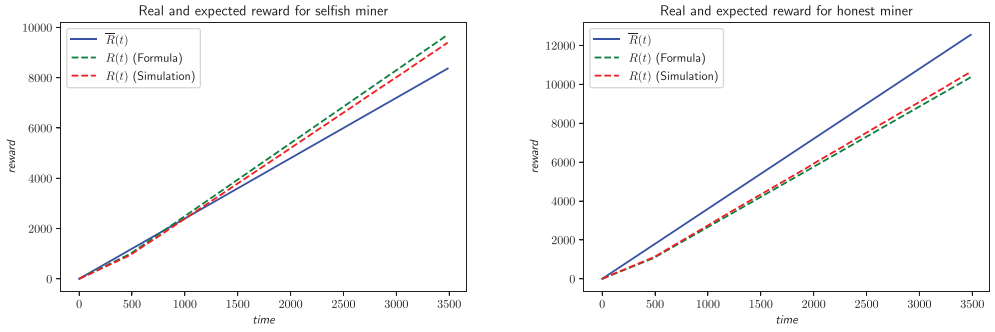


Fig. 8. Value of T_1 for different values of α.



(a) $\alpha = 0.30$ (b) $\alpha = 0.45$

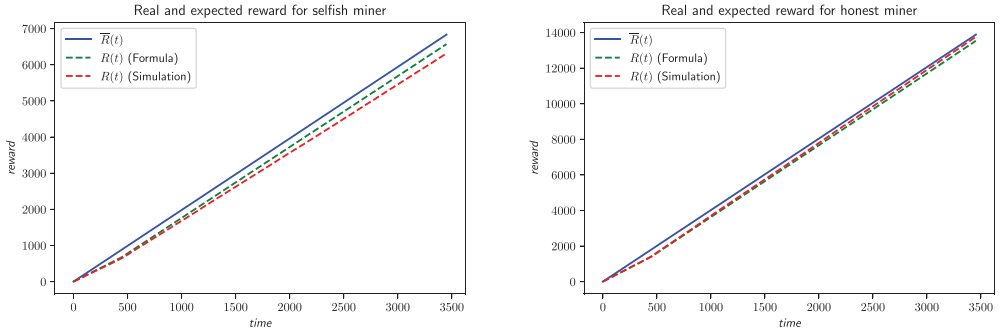
Fig. 9. *D* for different values of α.



(a) Selfish miner reward (b) Honest miner reward

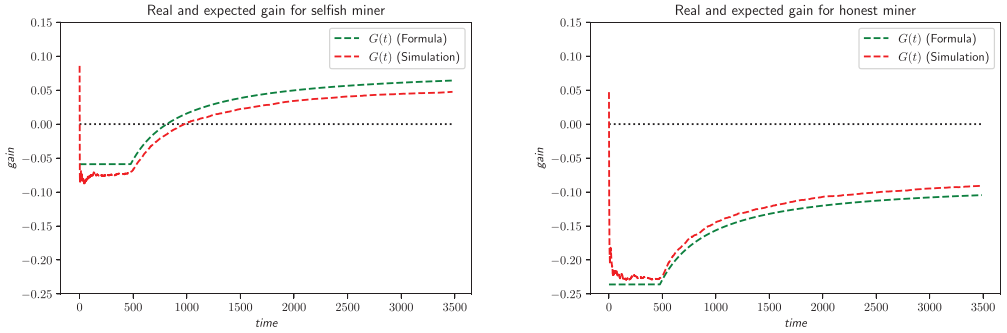
Fig. 10. Rewards for miners when $\alpha = 0.40$.

Now we can 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.



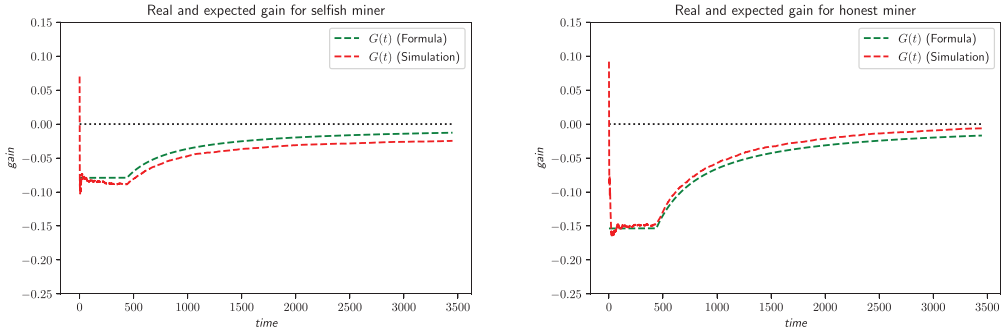
(a) Selfish miner reward (b) Honest miner reward

Fig. 11. Rewards for miners when $\alpha = 0.33$.



(a) Selfish miner gain (b) Honest miner gain

Fig. 12. Gain for miners when $\alpha = 0.40$.

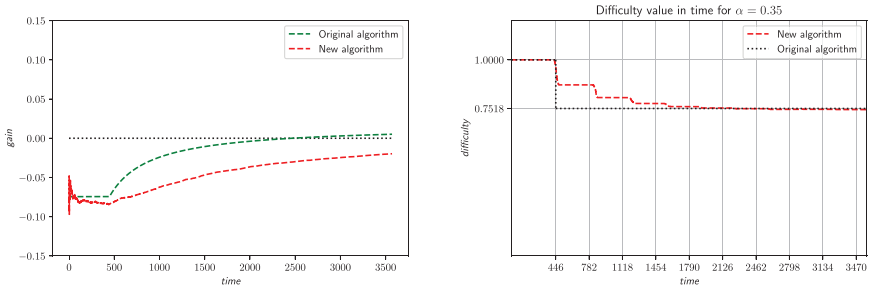


(a) Selfish miner gain (b) Honest miner gain

Fig. 13. Gain for miners when $\alpha = 0.33$.

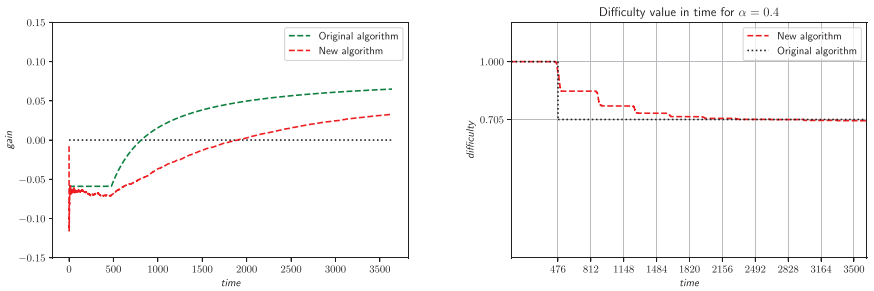
6.4.2. Discussion on Q_1

To answer the first research question, Q_1, we analyze the gain for selfish mining in different configurations. Figs. 14,15, 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.



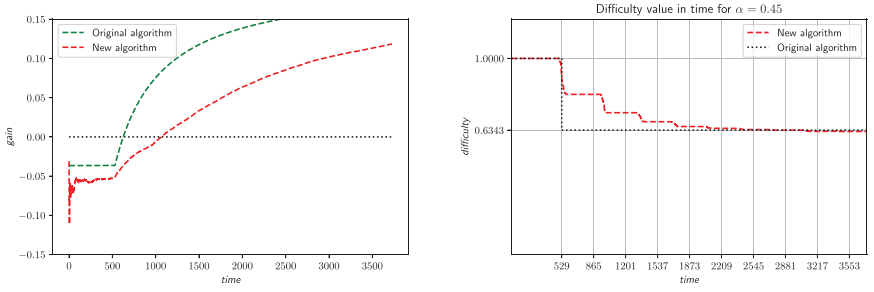
(a) Selfish miner gain (b) Network difficulty

Fig. 14. Comparison of different DAAs ($\alpha = 0.35$).



(a) Selfish miner gain (b) Network difficulty

Fig. 15. Comparison of different DAAs ($\alpha = 0.40$).



(a) Selfish miner gain (b) Network difficulty

Fig. 16. Comparison of different DAAs ($\alpha = 0.45$).

As shown in Figs. 14a,15a,16a, 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 zero line) goes further in time, which makes the non-profitable period of selfish mining longer.

To better understand the effect of Zeno's DAA over a larger range of α values, we summarize the results in Fig. 17. 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.

In Figs. 14b,15b,16b, 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 abruptly and in one step.

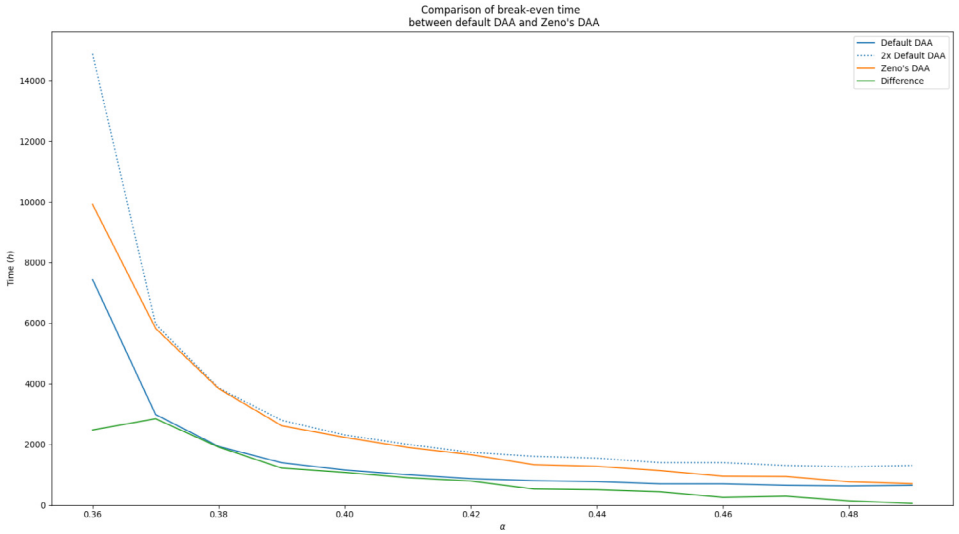
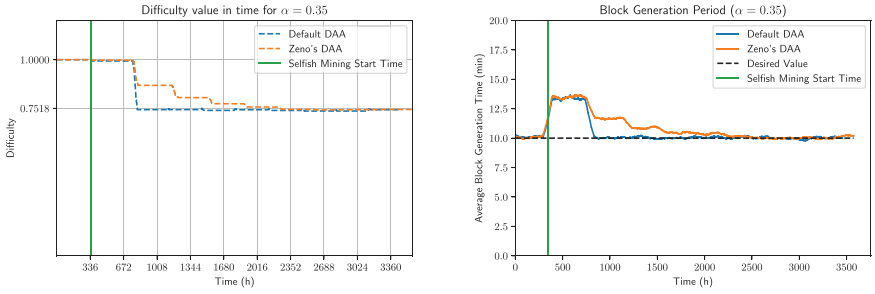


Fig. 17. Comparison of break-even times between default DAA and Zeno's DAA over a range of α.



(a) Difficulty Value (b) Block Generation Time

Fig. 18. Comparison between two DAAs.

The analysis above covers Q_1 from our research questions in Section 6.1 which is related to P_1 from the list of properties of an alternative DAA that is mentioned in Section 5 (Page 19).

6.4.3. 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 2.3. This means by using Zeno's DAA, we improve the security of the Bitcoin 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.

Fig. 18 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 (shown in the green vertical line). Fig. 18a shows the comparison of the difficulty values for the two DAAs. This figure is similar to Fig. 14b. Fig. 18b shows the average block generation time for 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 increases 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.

7. Discussion

In this section, we discuss how our proposed method satisfies the rest of the properties of a good DAA, P_3 to P_5, and also we a few other characteristics of Zeno's DAA in terms of defence and applicability.

7.1. 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 to a lower block generation rate, 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.

7.2. 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 (Eq. 13) to the computations. This equation is trivial to implement and adds an extremely insignificant and negligible computational load to the network.

7.3. 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 Eq. 11), 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 Eq. 13), 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.

7.4. 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. Instead, 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.

7.5. Implementing Zeno's DAA on Bitcoin or Another Existing Blockchain

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 (Eq. 13), this change is not backward compatible because, in case of an increase in the difficulty, the acceptable difficulty value in Zeno's DAA is lower than Bitcoin's default DAA. Therefore, 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.

8. Conclusion and Future Work

In this paper, we started with an introduction to Bitcoin and privacy and security issues concerning Bitcoin. Then we focused on a specific type of attack against mining pools, namely, selfish mining. We discussed two opposing ideas regarding the profitability of selfish mining. Eyal and Sirer [12] analyzed selfish mining and concluded that it is profitable in many cases. On the other hand, Wright and Savanah [46] argued that selfish mining is never profitable. As our first contribution, we showed that both scenarios are feasible if and when the important *time* factor is considered. Selfish mining always starts with a loss in revenue, similar to what Wright and Savanah showed in their analysis. Only after the difficulty adjustment, it starts to increase the revenue of the selfish miner until it surpasses the minimum required value for becoming profitable and finally converges to the final value that Eyal and Sirer predicted. We empirically showed this in Section 5.

On this basis, we identified the difficulty adjustment algorithm to be an important factor that affects the profitability of selfish mining, and as such, we analyzed the revenue and *gain* of the selfish miner with respect to *time* to see the effect time on the feasibility of selfish mining. For doing so, we introduce a simulator that we implemented to show the effect of selfish mining on the revenue of the miner under different setups.

To address the challenges posed by selfish miners on the default difficulty adjustment algorithm of Bitcoin, as our second contribution, we proposed an alternative difficulty adjustment algorithm, namely *Zeno's DAA*, which takes the history of difficulty into account. By analyzing our proposed method, we showed that our proposed method has the requirements of a suitable alternative for Bitcoin's DAA, and could discourage selfish behaviour by extending the waiting time before selfish mining becomes profitable.

We also empirically evaluated our proposed method in Section 6. Our results show that our proposed DAA, although simple, could effectively extend the waiting period to more than double in every setting while keeping the block generation rate within a reasonable margin.

However, this is not a perfect alternative for the original difficulty adjustment. For example, one of the problems with our proposed Zeno's DAA is that it acts symmetric, meaning it increases and decreases the difficulty in the same way. However, to better maintain the scalability, a DAA may need to be more sensitive to increasing difficulty than decreasing it. Another point is that a DAA could be parametric, meaning that it could be customizable by the needs of the network. We plan to extend this algorithm and propose better difficulty adjustment algorithms to overcome these issues. As our future works, we are planning to work on more sophisticated alternative DAAs that could potentially improve the performance of Zeno's DAA in terms of extending the profitability time, maintaining a constant block generation rate and allowing better scalability for the network. Also, we plan to perform a mathematical analysis for our proposed method to backup our simulation analysis.

CRediT authorship contribution statement

Hamid Azimy: Conceptualization, Methodology, Software, Writing - original draft. **Ali A. Ghorbani:** Supervision, Conceptualization, Methodology, Writing - review & editing, Funding acquisition. **Ebrahim Bagheri:** Supervision, Conceptualization, Methodology, Writing - review & editing.

Data availability

Data will be made available on request.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] E. Androulaki, G.O. Karame, M. Roeschlin, T. Scherer, S. Capkun, Evaluating user privacy in bitcoin, in: International Conference on Financial Cryptography and Data Security, Springer, 2013, pp. 34–51.
[2] H. Azimy, A. Ghorbani, Competitive selfish mining, in: 2019 17th International Conference on Privacy, Security and Trust (PST), IEEE, 2019, pp. 1–8.
[3] L. Bahack, Theoretical bitcoin attacks with less than half of the computational power (draft), 2013. arXiv preprint arXiv:1312.7013.
[4] D. Berdik, S. Otoum, N. Schmidt, D. Porter, Y. Jararweh, A survey on blockchain for information systems management and security, Information Processing & Management 58 (2021) 102397.
[5] J. Bonneau, A. Narayanan, A. Miller, J. Clark, J.A. Kroll, E.W. Felten, Mixcoin: Anonymity for bitcoin with accountable mixes, in: International Conference on Financial Cryptography and Data Security, Springer, 2014, pp. 486–504.
[6] V. Chicarino, C. Albuquerque, E. Jesus, A. Rocha, On the detection of selfish mining and stalker attacks in blockchain networks. Annals of Telecommunications, 2020. pp. 1–10.
[7] M. Conti, E.S. Kumar, C. Lal, S. Ruj, A survey on security and privacy issues of bitcoin, IEEE Communications Surveys & Tutorials 20 (2018) 3416–3452.
[8] N.T. Courtois, L. Bahack, On subversive miner strategies and block withholding attack in bitcoin digital currency, 2014. arXiv preprint arXiv:1402.1718.
[9] M. Davidson, T. Diamond, et al, On the profitability of selfish mining against multiple difficulty adjustment algorithms, IACR Cryptol. ePrint Arch. 2020 (2020) 94.

[10] C. Decker, R. Wattenhofer, Information propagation in the bitcoin network, in: Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on, IEEE, 2013, pp. 1–10.
[11] I. Eyal, E.G. Sirer, Bitcoin is broken, 2013. [Online; Accessed Feb. 2020].
[12] I. Eyal, E.G. Sirer, Majority is not enough: Bitcoin mining is vulnerable, in: International conference on financial cryptography and data security, Springer, 2014, pp. 436–454.
[13] Grunspan, C., & Pérez-Marco, R. (2018). On profitability of selfish mining. arXiv preprint arXiv:1805.08281.
[14] E. Heilman, One weird trick to stop selfish miners: Fresh bitcoins, a solution for the honest miner, in: In International Conference on Financial Cryptography and Data Security, Springer, 2014, pp. 161–162.
[15] Heilman, E., Kendler, A., Zohar, A., & Goldberg, S. (2015). Eclipse attacks on bitcoin's peer-to-peer network. In 24th {USENIX}) Security Symposium ({USENIX}) Security 15) (pp. 129–144).
[16] V. Holotescu, R. Vasiu, Challenges and emerging solutions for public blockchains. BRAIN, Broad Research in Artificial Intelligence and Neuroscience 11 (2020) 58–83.
[17] D. Hopwood, S. Bowe, T. Hornby, N. Wilcox, Zcash protocol specification, GitHub, San Francisco, CA, USA, 2016.
[18] K. Huang, X. Zhang, Y. Mu, F. Rezaeibagha, X. Du, Scalable and redactable blockchain with update and anonymity, Information Sciences 546 (2021) 25–41.
[19] M.C.K. Khalilov, A. Levi, A survey on anonymity and privacy in bitcoin-like digital cash systems, IEEE Communications Surveys & Tutorials 20 (2018) 2543–2585.
[20] Y. Kwon, D. Kim, Y. Son, E. Vasserman, Y. Kim, Be selfish and avoid dilemmas: Fork after withholding (faw) attacks on bitcoin, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2017, pp. 195–209.
[21] J. Li, J. Wu, L. Chen, Block-secure: Blockchain based scheme for secure p2p cloud storage, Information Sciences 465 (2018) 219–231.
[22] L. Li, J. Liu, X. Chang, T. Liu, J. Liu, Toward conditionally anonymous bitcoin transactions: A lightweight-script approach, Information Sciences 509 (2020) 290–303.
[23] P. Li, H. Xu, T. Ma, An efficient identity tracing scheme for blockchain-based systems, Information Sciences 561 (2021) 130–140.
[24] Y. Liu, Y. Hei, T. Xu, J. Liu, An evaluation of uncle block mechanism effect on ethereum selfish and stubborn mining combined with an eclipse attack, IEEE Access 8 (2020) 17489–17499.
[25] Maxwell, G. (2013). Coinjoin: Bitcoin privacy for the real world. [Online; Accessed Feb. 2021].
[26] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G.M. Voelker, S. Savage, A fistful of bitcoins: characterizing payments among men with no names, in: Proceedings of the 2013 conference on Internet measurement conference, 2013, pp. 127–140.
[27] D. Meshkov, A. Chepurnoy, M. Jansen, Short paper: Revisiting difficulty control for blockchain systems, in: Data Privacy Management, Cryptocurrencies and Blockchain Technology, Springer, 2017, pp. 429–436.
[28] I. Miers, C. Garman, M. Green, A.D. Rubin, Zerocoin: Anonymous distributed e-cash from bitcoin, in: Security and Privacy (SP), 2013 IEEE Symposium on, IEEE, 2013, pp. 397–411.
[29] S.G. Motlagh, J. Mišić, V.B. Mišić, Analysis of selfish miner behavior in the bitcoin network, in: ICC 2021-IEEE International Conference on Communications, IEEE, 2021, pp. 1–6.
[30] S.G.G. Motlagh, J. Misic, V.B. Misic, The impact of selfish mining on bitcoin network performance, in: IEEE Transactions on Network Science and Engineering, 2021.
[31] K. Nayak, S. Kumar, A. Miller, E. Shi, Stubborn mining: Generalizing selfish mining and combining with an eclipse attack, in: 2016 IEEE European Symposium on Security and Privacy (EuroS&P), IEEE, 2016, pp. 305–320.
[32] K.A. Negy, P.R. Rizun, E.G. Sirer, Selfish mining re-examined, in: International Conference on Financial Cryptography and Data Security, Springer, 2020, pp. 61–78.
[33] Niu, J., & Feng, C. (2019). Selfish mining in ethereum. arXiv preprint arXiv:1901.04620.
[34] W. Ren, J. Hu, T. Zhu, Y. Ren, K.-K.R. Choo, A flexible method to defend against computationally resourceful miners in blockchain proof of work, Information Sciences 507 (2020) 161–171.
[35] Rosenfeld, M. (2011). Analysis of bitcoin pooled mining reward systems. arXiv preprint arXiv:1112.4980.
[36] T. Ruffing, P. Moreno-Sanchez, A. Kate, Coinshuffle: Practical decentralized coin mixing for bitcoin, in: European Symposium on Research in Computer Security, Springer, 2014, pp. 345–364.
[37] M. Saad, L. Njilla, C. Kamhoua, A. Mohaisen, Countering selfish mining in blockchains, in: 2019 International Conference on Computing, Networking and Communications (ICNC), IEEE, 2019, pp. 360–364.
[38] Saad, M., Spaulding, J., Njilla, L., Kamhoua, C., Shetty, S., Nyang, D., & Mohaisen, A. (2019b). Exploring the attack surface of blockchain: A systematic overview. arXiv preprint arXiv:1904.03487.
[39] A. Sapirshtein, Y. Sompolinsky, A. Zohar, Optimal selfish mining strategies in bitcoin, in: International Conference on Financial Cryptography and Data Security, Springer, 2016, pp. 515–532.
[40] E.B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, M. Virza, Zerocash: Decentralized anonymous payments from bitcoin, in: 2014 IEEE Symposium on Security and Privacy (SP), IEEE, 2014, pp. 459–474.
[41] Solat, S., & Potop-Butucaru, M. (2016). Zeroblock: Preventing selfish mining in bitcoin. arXiv preprint arXiv:1605.02435.
[42] L. Valenta, B. Rowan, Blindcoin: Blinded, accountable mixes for bitcoin, in: International Conference on Financial Cryptography and Data Security, Springer, 2015, pp. 112–126.
[43] Viswanathan, S., & Shah, A. (2018). Scalability trilemma. [Online; Accessed Dec. 2021].
[44] Wang, C., Chu, X., & Yang, Q. (2019). Measurement and analysis of the bitcoin networks: A view from mining pools. arXiv preprint arXiv:1902.07549,.
[45] H. Wang, Q. Yan, V.C. Leung, The impact of propagation delay to different selfish miners in proof-of-work blockchains, Peer-to-Peer Networking and Applications (2021) 1–8.
[46] Wright, C.S., & Savanah, S. (2017). The fallacy of the selfish miner in bitcoin: An economic critique. Available at SSRN: https://ssrn.com/abstract=3009466 [Online; Accessed Nov. 2018].
[47] Yang, R., Chang, X., Mišić, J., & Mišić, V.B. (2021). Deep-dive analysis of selfish and stubborn mining in bitcoin and ethereum. arXiv preprint arXiv:2112.02588.
[48] S. Zhang, K. Zhang, B. Kemme, A simulation-based analysis of multiplayer selfish mining, in: 2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), IEEE, 2020, pp. 1–5.
[49] W. Zhang, Z. Zhang, H.-C. Chao, M. Guizani, Toward intelligent network optimization in wireless networking: An auto-learning framework, IEEE Wireless Communications 26 (2019) 76–82.
[50] W. Zhang, Z. Zhang, S. Zeadally, H.-C. Chao, V.C. Leung, Masm: A multiple-algorithm service model for energy-delay optimization in edge artificial intelligence, IEEE Transactions on Industrial Informatics 15 (2019) 4216–4224.