University of London
Imperial College of Science, Technology and Medicine
Department of Computing

Fairer, Faster, More Foreseeable: Incentives, Throughput and Stability of Proof of Work Blockchains

Sam Maximilian Werner

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of the University of London and
the Diploma of Imperial College, July 2023

Abstract

Blockchains employ internal and external incentive structures to influence participant behaviour, maintain network security, and ensure stable throughput. Internal incentives, like block rewards and transaction fees, are embedded within the blockchain design, while external incentives arise from market forces and competition. Both incentive structures are crucial for shaping blockchain behaviour and network efficiency.

The primary motivation of this thesis is to examine how misaligned incentive structures can negatively affect stability in Proof-of-Work blockchains, focusing on stable block and transaction throughput. The thesis aims to provide novel insights into the negative impact of unstable throughput on individual agents and the network as a whole. Additionally, it evaluates potential attack vectors resulting from misconstructed incentive structures, past exploits, and proposes fairer and more robust mechanisms to align incentives, ensuring higher throughput stability and network security.

The contributions of this thesis include the development of an open-source simulation framework called *PoolSim*. It enables the analysis of miner behaviour under different mining pool reward distribution schemes, including the profitability evaluation of queue-based manipulation strategies and pool-hopping between such pools. The thesis introduces the *uncle traps* attack, specific to Ethereum queue-based mining pools, which adversely affects block throughput and presents a fix to the uncle block reward distribution mechanism.

Furthermore, this thesis examines the impact of difficulty adjustment algorithms on block throughput. It identifies instability in block solve times due to cyclicality observed in Bitcoin Cash and analyses how miners' behaviour contributes to this phenomenon. A novel difficulty algorithm based on a negative exponential filter is derived, eliminating oscillations and ensuring more stable block solve times.

Lastly, the thesis addresses transaction throughput improvement by presenting a gas price prediction model for Ethereum. It combines deep-learning-based price forecasting with an urgency-based algorithm, optimising the trade-off between timely inclusion and transaction cost. Empirical analysis and real-world evaluation demonstrate significant cost savings with minimal delays compared to existing mechanisms.

Copyright

Acknowledgements

I would like to express my thanks and appreciation to:

- I would like to express my deepest gratitude to my PhD supervisor, Professor William Knottenbelt, for his unwavering support and guidance throughout my academic journey. From my Master thesis to the pursuit of a PhD, Will has been a constant source of encouragement, always fostering a positive and enjoyable research environment. His expertise and mentorship have been invaluable, and I am truly grateful for his presence in my life.

- I would also like to extend my appreciation to the Brevan Howard Centre for Financial Analysis for providing the funding that made my PhD research possible. Their generous support has allowed me to delve into the realm of blockchain technology, decentralised finance and explore innovative ideas in those fields.

- I owe a special debt of gratitude to my collaborators from the office ACE358. Working alongside them has been an enriching experience, filled with camaraderie and a shared passion for our research. Their presence has made the journey more enjoyable, and I am grateful for their constant support.

- I would like to acknowledge Alexei Zamyatin for his instrumental role in inspiring me to pursue a PhD and delve into the captivating world of blockchain and cryptocurrency. His encouragement and enthusiasm have been pivotal in shaping my research interests and expanding my horizons. I would also like to thank Dragos Ilie and Iain Stewart for the countless late-night discussions we had in Will's office. Their insights and intellectual camaraderie have greatly enriched my academic journey.

- To my dear friends and collaborators, Daniel Perez, Lewis Gudgeon, and Paul Pritz, I extend my heartfelt appreciation. Our collaborations on various projects have been intellectually stimulating and rewarding. Among many things, Daniel's technical guidance, Lewis's introduction to the world of Decentralised Finance, and Paul's curiosity about queue-based mining pools have all been instrumental in shaping my research journey. Moreover, the countless long nights of laughter have provided me with the strength to overcome challenges. I am truly grateful for their friendship.

- Finally, I want to express my profound gratitude to my parents and siblings for their unwavering belief in me. Their constant support, encouragement, and love have been my pillar of strength throughout this journey. I am deeply grateful for their presence in my life and their belief in me has played a pivotal role in my achievements.

Dr. Ellie Sattler: “So, what are you thinking?”
Dr. Alan Grant: “We’re out of a job.”
Dr. Ian Malcolm: “Don’t you mean extinct?”

Jurassic Park, 1993

Contents

List of Tables

List of Figures

Listings

Chapter 1

Introduction

1.1 Motivation

In the context of blockchains, internal (or embedded) incentive mechanisms are incentives that are built into the design of the blockchain itself, which influence the behaviour of participants in the network. These mechanisms can include block rewards, transaction fees, and other incentives that motivate miners to contribute their computing power to the network and maintain the security and integrity of the blockchain. External incentive mechanisms, on the other hand, are incentives that are not built into the design of the blockchain, but rather come from outside the network. These can include market forces, competition between mining pools, and the availability of alternative blockchains. External incentives can influence the behaviour of miners and other participants in the network, but they are not directly controlled by the blockchain protocol itself. Overall, both internal and external incentive mechanisms play a vital role in shaping the behaviour of participants in blockchain networks and affecting the overall security and efficiency of the network. By understanding and optimising these incentives, blockchain designers can create more robust and stable networks.

Key measures of stability in Proof-of-Work blockchains are block and transaction throughput. Stable block throughput is critical for Proof-of-Work blockchains because it ensures the reliability and efficiency of the network. Block throughput refers to the number of blocks that

can be added to the blockchain in a given time period, which directly affects the transaction processing capacity of the network. If block throughput is unstable, it can cause delays in transaction confirmations (transaction throughput), which in turn can lead to higher transaction fees, longer wait times, and ultimately decreased user satisfaction. Additionally, unstable block throughput can make the network vulnerable to attacks, such as denial-of-service attacks, and can make it more difficult for the network to maintain consensus. Therefore, stable block throughput is critical for the overall functionality and security of Proof-of-Work blockchains.

The primary motivation of this thesis is to explore how misaligned incentive structures can directly impact the stability of Proof-of-Work blockchains. We shall define stability in the context of stable block and transaction throughput. The incentive mechanisms investigated in this thesis will be both external and internal mechanisms of Proof-of-Work chains. By doing so, we seek to provide novel insights on how both individual agents and the network itself may be negatively affected from unstable throughput. Furthermore, we aim to (i) evaluate the extent to which such misalignment may result in attack vectors, (ii) evaluate the degree to which such vectors have been exploited in the past, and lastly, (iii) introduce more robust mechanisms that align incentives for participating agents to ensure overall throughput stability and network security.

1.2 Objectives

The following points outline the objectives of this thesis.

Objective 1: Development of an Extensible Mining Pool Simulation Framework. For this objective, the aim is to develop an open-source, extensible, discrete event-based simulation framework. Previous work on mining pools (e.g., [ZWW+17]) relied on event simulations to identify novel attack strategies under different mining pool reward schemes. However, there is a lack of extensible and open-source mining pool simulation tools. Hence, an objective is to develop an extensible framework that facilitates research on mining pools and allows academics to validate their hypotheses through simulations at ease.

Objective 2: Investigate the Manipulation of Mining Pool Reward Schemes. While the research on the fairness of traditional mining pool rewards schemes is quite extensive (e.g.,[Ros11]), the authors of [ZWW+17] have demonstrated that some mining pool reward schemes can still (rather easily) become subject to manipulation. Specifically, queue-based mining pools are yet to be proven to be fair and have shown to offer attack vectors whereby miners can increase their profits at the cost of others. Hence, an objective of this thesis is to explore the extent to which there are further attack vectors rooted in mining pools and whether these could result in negative effects in terms of network stability (e.g., lower block rate, lower profits for miners).

Objective 3: Examine the Robustness of Internal Stability Mechanisms in PoW Blockchains. This objective investigates the impact of internal mechanisms that are employed by PoW blockchains to ensure stability in terms of block throughput. This may include mechanisms, such as the difficulty adjustment algorithm or internal incentive structures, such as the block reward and transaction fees. The aim is to examine the extent to which such mechanisms are robust with respect to miner behaviour and whether these can be manipulated at the cost of other network participants (e.g., other users that may suffer from lower block throughput).

Objective 4: Ensuring Reliable Transaction Throughput in PoW Blockchains. This objective specifically focuses on improving transaction throughput in Proof-of-Work blockchains. The most common way of ensuring that a transaction gets confirmed (i.e., included in a valid block) is to set an appropriate transaction fee. However, the tools and mechanisms that exist to recommend appropriate fees vary greatly by design. Oftentimes, the sender of a transaction ends up overpaying for their transaction to be included in a block, as the risk of the transaction not getting included is simply too high. This is a major issue from a usability point of view as transaction throughput can be rather unreliable if recommendation mechanisms are not accurate. The objective is to examine the existing tools that are used for recommending transaction fees and to explore alternative, more accurate mechanisms.

1.3 Contributions

This thesis investigates the effects of misaligned incentive mechanisms in the context of Proof-of-Work blockchains. These mechanisms can lead to economic disadvantages for individual agents, such as miners, as well as to severe instabilities in block and transaction throughput. The thesis examines both external and internal incentive mechanisms of Proof-of-Work blockchains and presents empirical findings that are validated through simulations. Additionally, the thesis proposes new mechanisms that would eliminate the discussed vulnerabilities in throughput caused by misaligned incentives.

Specifically, the following contributions have been made as part of this thesis:

***PoolSim*: A Mining Pool Simulation Framework.** We first present an open-source, extensible, discrete event-based simulation framework called *PoolSim*, which can be used to simulate miner behaviour under different mining pool reward distribution schemes. We extend existing research on the manipulation of queue-based mining pool reward distribution schemes [ZWW+17], by evaluating the profitability of different queue-based manipulation strategies. Furthermore, we present the first simulation of pool-hopping strategies between two queue-based mining pools and evaluate the profitability of this approach.

***Uncle Traps*: Exploiting Queue-based Mining Pools.** After simulating different mining strategies in queue-based mining pools, we found that an adversarial miner in the pool is able to increase their share of earnings at the cost of other miners in the queue-based pool. We term this specific attack "uncle traps". This attack was previously not formally studied and only applies to Ethereum queue-based mining pools. We provide empirical evidence of how miners have employed uncle traps to increase their own profits at the cost of other miners' earnings in the pool, and consequently caused lower block throughput in the network. We simulate how uncle traps can be utilised in a profitable manner and how profits can be further increased should the adversary additionally employ a share-withholding strategy. Lastly, we propose a novel fix to the uncle block reward distribution mechanism used by queue-based mining pools, which would make the uncle traps attack no longer feasible.

Unstable Block Solve Times and Broken Difficulty Algorithms. In order to understand how not only broken external incentive mechanisms can cause a reduction in block throughput, we subsequently examine how the design of internal blockchain stability mechanisms can have a similar, more severe effect. Specifically, we examine how difficulty adjustment algorithms can impact block throughput in Proof-of-Work blockchains. First, we provide empirical evidence showing that the design of difficulty algorithms can lead to unstable block solve times, as evidenced by the cyclicality observed in Bitcoin Cash. Second, we show how miners' behaviour can contribute to this phenomenon over time through coin-hopping strategies. To address this issue, we derive a new difficulty algorithm based on a negative exponential filter that prevents positive feedback loops and offers additional benefits such as history agnosticism. We demonstrate through a simulated mining environment that this new algorithm can eliminate oscillations in block solve times as observed in Bitcoin Cash. Finally, we suggest that other Proof-of-Work blockchains could apply this new difficulty algorithm and ensure stable block throughput.

A Better Gas Price Recommendation Mechanism for Ethereum. Lastly, after having shown how block throughput can become unstable due to broken external and internal incentive mechanisms, we focus on how transaction throughput can be improved. We present a novel gas price prediction model to increase the likelihood of a transaction being included in a block in Ethereum. The mechanism addresses the trade-off between timely inclusion and transaction cost by using a deep-learning-based price forecasting model combined with an algorithm that employs a user-specific urgency value to recommend gas prices. We present an empirical analysis of historical block data, followed by a predictive model used for the recommendation of gas prices. We then evaluate the proposed mechanism on real-world data and demonstrate that it results in significant cost savings of over 50% while only incurring a delay of 1.3 blocks compared to the gas price recommendation mechanism of the most widely used Ethereum client.

1.4 Statement of Originality

I declare that this thesis was composed by myself, and that the work that it presents is my own, as well as joint work with co-authors of the included publications (see Section 1.5). All work presented in this thesis has been solely submitted as part of this thesis and not as part of any other degree or qualification, neither by myself nor any of the co-authors.

1.5 Publications

The research presented in this thesis relies on the following peer-reviewed publications[1]:

- **Chapter 3.** Sam M Werner and Daniel Perez. PoolSim: A Discrete-Event Mining Pool Simulation Framework. In *Mathematical Research for Blockchain Economy: 1st International Conference MARBLE 2019, Santorini, Greece*, pages 167–182. Springer, 2020

- **Chapter 4.** Sam M Werner, Paul J Pritz, Alexei Zamyatin, and William J Knottenbelt. Uncle Traps: Harvesting Rewards in a Queue-based Ethereum Mining Pool. In *Proceedings of the 12th EAI International Conference on Performance Evaluation Methodologies and Tools*, pages 127–134, 2019

- **Chapter 5.** Dragos I Ilie, Sam M Werner, Iain D Stewart, and William J Knottenbelt. Unstable Throughput: When the Difficulty Algorithm Breaks. In *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–5. IEEE, 2021

- **Chapter 6.** Sam M Werner, Paul J Pritz, and Daniel Perez. Step on the Gas? A Better Approach for Recommending the Ethereum Gas Price. In *Mathematical Research for Blockchain Economy: 2nd International Conference MARBLE 2020, Vilamoura, Portugal*, pages 161–177. Springer, 2020

[1] Content taken from these publications also includes parts from pre-prints of these publications.

Please note that parts of the background section have also been taken from the aforementioned publications.

The following publications are not included in this thesis, yet are a result of research conducted throughout the course of the PhD:

- Sam M Werner, Daniel Perez, Lewis Gudgeon, Ariah Klages-Mundt, Dominik Harz, and William J Knottenbelt. SoK: Decentralized Finance (DeFi). In *AFT '22: Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, pages 30–46. ACM, 2022
- Daniel Perez, Sam M Werner, Jiahua Xu, and Benjamin Livshits. Liquidations: DeFi on a Knife-edge. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part II 25*, pages 457–476. Springer, 2021
- Lewis Gudgeon, Sam Werner, Daniel Perez, and William J Knottenbelt. DeFi Protocols for Loanable Funds: Interest rates, Liquidity and Market Efficiency. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 92–112, 2020

1.6 Dissertation Outline

This section outlines the remainder of this thesis.

Chapter 1: Introduction.

The introductory chapter provides a comprehensive overview of the research problem, objectives, and contributions of the thesis. It sets the context for the subsequent chapters by establishing the importance of investigating misaligned incentive mechanisms in Proof-of-Work blockchains.

Chapter 2: Background and Related Work.

This chapter delves into the foundational understanding of Proof-of-Work (PoW) blockchains, offering a thorough background on their underlying mechanisms and relevant concepts. It covers essential topics such as blockchain components, PoW consensus, forks, and incentives, providing the necessary groundwork for the subsequent chapters' discussions.

Chapter 3: PoolSim: Simulating Miner Behaviour in Mining Pools.

In this chapter, the thesis introduces *PoolSim*, an open-source and extensible simulation framework designed for modeling miner behavior in various mining pool reward distribution schemes. It explains the discrete event-based simulation approach used in PoolSim and evaluates the profitability of different queue-based manipulation strategies. Additionally, the chapter explores the simulation of pool-hopping strategies, contributing to a deeper understanding of mining pool dynamics.

Chapter 4: Misaligned Mining Incentives in Queue-based Mining Pools.

The focus of this chapter is the proposal and investigation of the "uncle trap" attack within Ethereum queue-based mining pools. Through empirical analysis, the chapter reveals how miners exploit uncle traps to increase their profits while adversely affecting other miners' earnings and block throughput in queue-based mining pools. The chapter further explores the simulation of profitable utilisation of uncle traps and proposes a novel fix to the uncle block reward distribution mechanism, mitigating the vulnerability of the attack.

Chapter 5: (Un)Stable Block Throughput: Feedback Loops and Difficulty Adjustment Algorithms.

Examining the impact of difficulty adjustment algorithms on block throughput in Proof-of-Work blockchains, this chapter presents empirical evidence of unstable block solve times, specifically the cyclicality observed in Bitcoin Cash. It analyses the contribution of miners' behavior, particularly coin-hopping strategies, to this phenomenon. To address the issue, we derive a difficulty algorithm based on a negative exponential filter, ensuring stable block solve times

and providing benefits such as history agnosticism.

Chapter 6: (Un)Stable Transaction Throughput: Leveraging the Cyclicality of Transaction Fees.

The objective of this chapter is to enhance transaction throughput in Ethereum by proposing a better gas price prediction model. It presents a deep-learning-based price forecasting model combined with an algorithm that incorporates user-specific urgency to recommend gas prices. The chapter includes an empirical analysis of historical block data, the development of a predictive model, and an evaluation of cost savings and transaction inclusion rates, demonstrating the efficacy of the proposed mechanism.

Chapter 7: Conclusion.

The concluding chapter summarises the achievements of the thesis and highlights the main contributions made. It discusses the implications of the findings and presents potential avenues for future research.

Chapter 2

Background and Related Work

In this chapter, we outline the core mechanisms of Proof-of-Work blockchains and explain how the Bitcoin decentralised consensus mechanism works.

2.1 The Rise of Blockchains

The Emergence of Bitcoin in Response to the 2008 Financial Crisis.

Bitcoin, the world's first decentralised digital currency, emerged in the wake of the 2008 global financial crisis. This crisis, characterized by bank failures, government bailouts, and a loss of trust in traditional financial systems, provided fertile ground for the development of an alternative financial system.

The original whitepaper titled "Bitcoin: A Peer-to-Peer Electronic Cash System" [Nak08] was published in October 2008 by an individual or group operating under the pseudonym Satoshi Nakamoto. It outlined a vision for a peer-to-peer electronic cash system that would operate without the need for intermediaries such as banks or government institutions. The whitepaper introduced the concept of a decentralized ledger, called the blockchain, to enable secure and transparent transactions by relying on a novel consensus mechanism that would allow distrusting participants to reach agreement over the true state of transactions.

Benefits of Distributed Ledgers in Comparison to Traditional Financial Systems.

The advent of Bitcoin brought attention to the benefits of distributed ledger technology (DLT) or blockchain technology. Distributed ledgers offer several advantages over traditional financial systems:

Decentralisation: Distributed ledgers are not controlled by a central authority, eliminating the need for intermediaries and reducing the risk of manipulation or censorship. This decentralised nature ensures transparency and trust in the system.

Security: Blockchain technology employs cryptographic techniques to secure transactions, making them tamper-resistant. The decentralised nature of the network, where multiple participants validate and verify transactions, enhances security and reduces the risk of fraud.

Efficiency: Traditional financial systems often involve complex processes and intermediaries, leading to delays and high transaction costs. Distributed ledgers can streamline and automate these processes, potentially resulting in faster and more cost-effective transactions.

Accessibility: Distributed ledgers enable financial inclusion by providing access to financial services for individuals who are unbanked or underbanked. With a smartphone and an internet connection, anyone can participate in the blockchain network and conduct transactions.

Satoshi Nakamoto and the Rise of Bitcoin and Distributed Ledger Technology.

The true identity of Satoshi Nakamoto remains unknown to this day. It is believed that Nakamoto's motivation was to create a decentralised currency that would eliminate the need for intermediaries and provide individuals with financial sovereignty.

The release of the Bitcoin whitepaper and the subsequent launch of the Bitcoin network in 2009 sparked a revolution in the world of finance. Bitcoin served as the catalyst for the rise of DLT and the development of numerous other blockchain platforms.

Following Bitcoin's success, various alternative cryptocurrencies emerged, each with its unique features and use cases. These cryptocurrencies, often referred to as altcoins, expanded the scope of DLT and facilitated the development of decentralised applications (DApps).

Today's Landscape of Bitcoin and Distributed Ledger Technology.

Bitcoin has come a long way since its inception. It has gained recognition as a legitimate asset class, with institutional investors and companies integrating it into their financial strategies. Additionally, the underlying blockchain technology has attracted interest from various industries beyond finance, including supply chain management, healthcare, and voting systems. Furthermore, the concept of distributed ledgers has expanded beyond public blockchains like Bitcoin. Private and permissioned blockchains have emerged, allowing enterprises to leverage the benefits of DLT within closed ecosystems while maintaining control over access and governance. However, challenges remain. Scalability, energy consumption, and regulatory frameworks are areas that continue to evolve as DLT matures. Efforts are being made to address these challenges through advancements in consensus algorithms, layer-two scaling solutions, and regulatory frameworks that balance innovation and consumer protection.

2.2 Blockchain Fundamentals

In this section, we outline the fundamental components of Proof-of-Work blockchains.

2.2.1 Peer-to-Peer Network

Proof-of-work blockchains rely on peer-to-peer (P2P) networks to ensure the integrity and security of the network. In a P2P network, each participant or node maintains a copy of the entire blockchain state, ensuring that there is no central point of failure or control. When a new transaction is initiated, it is broadcasted to all nodes in the network, and each node validates it by verifying its consistency with the network's rules and checking that the sender has sufficient funds to carry out the transaction. Nodes also broadcast newly created blocks containing previously unconfirmed transactions to the network, and all nodes in the network validate and add the block to their copy of the ledger. This distributed approach to blockchain verification ensures that no single entity has control over the network, making it secure, resilient,

and decentralised.

2.2.2 Transactions and Blocks

Blocks are data structures that make up the building blocks of a blockchain and contain a set of validated transactions. In addition to transactions, blocks also contain a *header* field, which contains information such as the block's height or the timestamp at which the block was produced. Each block is distinguished by a cryptographic hash that also references the previous block's hash, creating a secure and tamper-proof ledger that is distributed across a network of nodes. Each node in the network maintains a copy of the blockchain. While any participant in the network may propose a new block, the decision of which miner is granted the privilege to append a new block to the blockchain is determined by the underlying consensus algorithm.

2.2.3 Proof-of-Work

The underlying consensus mechanism of Bitcoin, termed *Nakamoto consensus*, builds on a random leader election process where participating nodes are required to invest computational power in solving cryptographically hard, memoryless and non-invertible puzzles. The latter is referred to as *Proof-of-Work* (PoW). The process of searching for solution candidates is termed *mining*, with participating nodes being referred to as *miners*. Miners create blocks by solving the PoW puzzle: trying different values for the nonce field of the block header, such that its `SHA-256` [PvW08] hash lies below a specified *target* value. The number of solution candidates, or hashes, a miner can compute per second denotes a miner's *hash rate*, typically expressed in megahashes[1] (MH/s) or gigahashes[2] per second (GH/s). Each miner that generates a valid solution to the PoW puzzle becomes the leader and is allowed to append the next block, and thus a set of previously unconfirmed transactions, to the blockchain.

In Bitcoin, the PoW is a partial pre-image attack on the SHA256 cryptographic hash function, where generating a valid solution via brute-forcing is hard, while verifying a hash against a

[1] 1 MH/s = 10^6 hashes per second
[2] 1 GH/s = 10^9 hashes per second

pre-image is easy.

$$\texttt{SHA256(SHA256(<block\_header>))} <= T \tag{2.1}$$

where T is a number between 0 and $2^{256}-1$ referred to as the *target*. The block header generally contains a nonce, namely a field that can be modified by miners to hold some arbitrary number such that equation 2.1 holds. By referencing the hash of the previous block header, the blocks get cryptographically chained together. Note that while the PoW function and the implementation of it may differ between PoW blockchains, the core design of referencing the hash of the previous block's header remains.

To incentivise honest participation and compensate miners for the computational effort invested, the miner of a block is rewarded a predefined amount of newly minted units of the underlying cryptocurrency, the *block reward*. For example, the block reward in Bitcoin started at 50 BTC per block but halves every four years. In addition to the block reward, the miner of a block receives all the fees of the transactions that are included in the block.

2.2.4 Difficulty

In PoW blockchains, the *difficulty* refers to the expected amount of hashes that a miner must perform, or the amount of "work" that needs to be done, in order to generate a valid solution to the PoW puzzle and thereby mine a block. The network difficulty is adjusted via a difficulty algorithm (DA) that controls transaction throughput by regulating the difficulty of generating PoW solutions based on the total computational power in the network, or the hash rate. Such algorithms are designed to ensure stable block times even during periods of hash rate oscillations, i.e., when miners join or leave the network. If the difficulty is not appropriate, it can cause varying response times for blockchain transactions, with either many blocks being processed in short periods of time or very few blocks being found in long periods. For instance, in Bitcoin, the difficulty is adjusted every 2,016 blocks (approx. every 2 weeks) based on the block solve times of the previous 2,016 blocks such that it is in line with the target average solve time of 10 minutes. A more detailed overview of DA design will be provided in Chapter 5.

2.2.5 Consensus and the Longest Chain Rule

The "longest chain rule" in PoW blockchains states that the valid chain with the most accumulated computational work (difficulty) or the longest chain, is the one that should be accepted by the network as the canonical blockchain. This is because the miner that finds the solution to the PoW puzzle and therefore gets to append a new block to the chain must have invested a significant amount of computational effort. Hence, their block should be considered the most valid and trustworthy, and the other miners in the network should add it to their copy of the blockchain. By following the longest chain rule, consensus is reached within the network, as all participants agree on the same version of the blockchain, ensuring its security and integrity.

2.3 Mining and Mining Pools

Mining remains a fundamental structural component for the effective workings of the decentralised consensus mechanism in Proof-of-Work cryptocurrencies, such as Bitcoin and Ethereum [But14]. Although some blockchains, such as Ethereum, have switched from a Proof-of-Work algorithm to a Proof-of-Stake mechanism, PoW remains as one of the most common consensus algorithms for cryptocurrencies. For instance, Bitcoin, the largest cryptocurrency by market capitalisation, still uses the PoW consensus algorithm, highlighting the continued relevance and importance of PoW in the cryptocurrency industry. However, rising difficulty levels of the cryptographic puzzles underpinning the mining process have posed severe constraints on the frequency of rewards paid to individuals trying to find PoW solutions. In this section, we provide a more detailed overview of cryptocurrency mining and mining pools, specifically with respect to the earnings of a miner. Furthermore, we present some of the most popular mining pool reward schemes used, as presented in [Ros11, ZWW+17].

2.3.1 Solo Mining

The simplest form of mining is called solo-mining and refers to a single miner mining independently by generating candidate solutions to the PoW puzzle. The discovery of a block with a constant hash rate, represented by h, conforms to a Poisson distribution, whose rate parameter is defined by $\lambda = \frac{h}{D}$. In this context, D refers to the difficulty level of the network[3]. Therefore, a miner who mines alone and has a hash rate of h and mines for a period of time t can anticipate a revenue, as represented by the equation: $E[R] = \frac{htB}{D}$, where B denotes the block reward and D is the network difficulty. The highest risk faced by solo miners is the financial risk caused by the high payout variance that they experience.

2.3.2 Mining Pools

A *mining pool* consists of a group of miners that pool their computational resources together to reduce payout variance. Mining pools are managed by a centralised operator who issues shares or PoW problems with a lower difficulty level than the network difficulty in order to track the amount of work performed by miners in a pool. Each share has a probability $\frac{d}{D}$ of being a valid solution to the network PoW, with d being the share difficulty. Consequently, miners in the pool submit valid shares to the operator, where each share is a valid candidate PoW solution. The pool operator's role is to verify whether a submitted share is a solution to the PoW puzzle. When a block gets mined by the pool (i.e., a share is a solution), the block reward (minus some operator fee) gets distributed among the pool participants according to some reward distribution scheme set by the pool operator.

In the following subsections, we outline the most common reward distribution schemes used by mining pools. We compare these schemes, by expressing the expected reward for a miner on a *per-round* basis, i.e., for the time period between two blocks mined by the pool.

[3]Note that the range of D is more of an implementation detail and is further discussed in Section 5.1.1.

Proportional Payout Scheme

The simplest and perhaps most intuitive mining pool reward scheme is the *proportional* reward allocation. In this scheme, block rewards are distributed proportionally between miners according to the number of shares each miner submitted for the current round. A *round* refers to the time period between two blocks being mined by a pool. Hence, if a miner submitted n shares during a round in which N shares have been submitted in total, the miner's payout would be equal to $\frac{n}{N}(1-f)B$, where f is the fee charged by the pool operator.

Pay-Per-Share (PPS)

In a *pay-per-share* scheme, the pool operator pays a miner $(1-f)pB$, where p is the probability that the share is a PoW solution, i.e. $\frac{d}{D}$. By doing so, the operator fully absorbs the miner's payout variance. Hence, an operator of a PPS pool could make profits on short rounds, while being exposed to high losses on long rounds. Typically, in order to compensate for this risk, PPS operators charge higher fees compared to other reward schemes. For understanding optimal reserve balances in a PPS pool we point the reader to Rosenfeld [Ros11], who formally examines this.

Pay-Per-Last-N-Shares (PPLNS)

Unlike many other traditional schemes, PPLNS abandons the concept of splitting rewards based on rounds, but rather distributes the block reward evenly among the last N shares submitted by miners, where N typically is a multiple of the network difficulty. This automatically takes away the incentive to only submit shares during the early period of a round. In a PPLNS pool, the expected reward normalised to a per round basis is found to be

$$E[R_i] = (1-f)B\frac{s_i}{N} \cdot \frac{N}{D} \tag{2.2}$$

where s_i is the number of shares submitted by miner i during the last N shares [Ros11].

Queue-based Payout Scheme (QB)

The *queue-based*[4] reward scheme was introduced by Ethpool, a small Ethereum mining pool. In a queue-based mining pool, miners receive a number of *credits* equal to the difficulty of a share for each submitted share. When a block is mined by the pool, the full block reward[5] is allocated to the miner in the pool with the highest accumulated credit balance, namely the *top miner* in a priority queue. Subsequently, the top miner's credit balance is reset to the difference between his and the second highest credit balance in the pool. The reason for not resetting the credits of a top miner to zero is to provide an incentive for a miner to continue to perform work for the pool once he reaches the top position in the queue, opposed to switching to some other pool.

Position	Miner	Credits
1	Bob	140
2	Alice	130
3	Carol	70

(a) Before Block i

Position	Miner	Credits
1	Alice	130
2	Carol	70
3	Bob	10

(b) After Block i

Position	Miner	Credits
1	Alice	140
2	Carol	75
3	Bob	20

(c) Before Block i+1

Position	Miner	Credits
1	Carol	75
2	Alice	65
3	Bob	20

(d) After Block i+1

Table 2.1: The priority queue with miners of different sizes over a series of blocks.

As first shown by [ZWW+17], a similar queue-based mining pool example is given in Table 2.1, showcasing that the credit resetting mechanism is non-uniform in the sense that the credits of top miners may be reset to differing balances. After a block has been mined by the pool (Table 2.1b), the top miner Bob has his credits reset to 10, the difference to Alice's credits. After each miner submits shares and receives credits, right before the next block is being mined (Table 2.1c), Alice is top of the queue. However, once the block has been mined (Table 2.1d), Alice is reset to a starting balance of 65 credits, a notably higher amount than Bob received. This

[4]Ethpool referred to this as a predictable solo mining pool, however, we shall employ the term "queue-based pool" as introduced in [ZWW+17]

[5]minus the pool operator fee

suggests that Bob would indeed have been better off had he allowed Alice to bypass him in the queue in order to receive a higher starting balance.

2.4 Incentive Structures in Blockchains

This section explores the concept of incentives in blockchain systems, both internal and external, and their role in shaping participant behavior, network security, and overall efficiency. It discusses the importance of aligned incentive structures for the stability and functionality of Proof-of-Work blockchains.

2.4.1 Internal Incentives in Proof-of-Work Blockchains

Internal incentives in Proof-of-Work (PoW) blockchains are the rewards and incentives that are inherent within the blockchain system itself. These incentives include block rewards and transaction fees that are embedded in the blockchain's design, serving as motivations for miners to participate in the mining process and validate transactions.

Examples of internal incentives include:

Block Rewards:. Miners are rewarded with a certain number of newly created cryptocurrency tokens for successfully mining and adding a new block to the blockchain.

Transaction Fees:. Miners also receive transaction fees paid by users for including their transactions in a block. These fees serve as an additional incentive for miners to prioritize and include transactions with higher fees.

2.4.2 External Incentives in Proof-of-Work Blockchains

External incentives in PoW blockchains arise from outside the network and are influenced by market forces and competition. These incentives can include the potential for financial

gain through the value appreciation of the cryptocurrency associated with the blockchain. Additionally, competition among miners and mining pools creates external pressures that drive participants to invest in more powerful hardware and improve mining strategies.

Examples of external incentives include:

Market Forces:. The value of the cryptocurrency associated with the blockchain can serve as an external incentive. As the value of the cryptocurrency increases, miners have a greater incentive to participate in mining to earn more rewards.

Competition:. The presence of other miners and mining pools competing for block rewards and transaction fees creates external pressure and incentives to invest in more powerful hardware, increase mining efficiency, and improve mining strategies.

Reputation and Network Effects:. Miners may also be driven by the desire to establish a positive reputation within the blockchain community. A good reputation can attract more users and investors to the blockchain network, increasing the value of the cryptocurrency and potentially leading to greater rewards.

Both internal and external incentives play crucial roles in shaping the behavior of participants in PoW blockchains. Internal incentives ensure the security and stability of the network by motivating miners to dedicate computational power to the mining process. External incentives provide economic and competitive motivations for miners to participate in the network and contribute to its functionality and security.

2.5 Network Forks

In Proof-of-Work blockchains, the main chain refers to the chain with the most accumulated performed work, i.e., the sum of difficulties of the mined blocks. A *fork* happens when at least two blocks reference the same past block. This may occur due to slow network propagation of blocks when two miners find a solution to the PoW at nearly the same time. As miners build

upon the block that reaches them first, the temporary fork is likely resolved after one branch exceeds the other in terms of work and becomes the new main chain. However, upgrades to the protocol rules can be performed either via *hard* or *soft* forks.

2.5.1 Hard Forks

A hard fork in a PoW blockchain refers to a permanent divergence from the current longest chain, resulting in two separate chains with different protocol rules. The majority of the nodes in the network will upgrade their software, such that the protocol rules are no longer compatible with the previous ones. Hence, newly created blocks will no longer be valid under the old protocol rules (the original chain), but will be valid on the new *forked* chain. An example of a hard fork would be a software upgrade that increases the maximum size of a block, as new blocks would no longer be backward compatible and thus valid under the old rules. Prominent hard forks of Bitcoin include Bitcoin Cash[Bit23] (Aug. 2017), Bitcoin SV[SV23] (November 2018), or Bitcoin Gold[Gol23] (Oct. 2017).

2.5.2 Soft Forks

A soft fork refers to a temporary divergence from the existing blockchain where only previously valid transactions and blocks become invalid. It occurs when a new rule or feature is added that is backward-compatible with the existing set of protocol rules. Unlike with a hard fork, nodes do not have to upgrade their software in order to be able to validate blocks. Nodes that keep the old software (rules) are still able to validate blocks, even under the newly introduced rules. An example of a soft fork would be a reduced block size, as new blocks would still be valid under the original set of protocol rules, thereby being backward compatible. A concept that expands on that of a soft fork is a so-called *velvet fork*, first introduced by [KMZ20] and further expanded on by [ZSJ+19]. A velvet fork offers a unique approach that doesn't rely on the majority support of participants and has the potential to prevent rule disagreement forks entirely. Unlike traditional forks, a velvet fork allows for the introduction of new protocol rules

without requiring consensus participants to upgrade. Under the velvet fork, blocks adhering to the new rules are considered valid under both the new and old rule sets. In essence, velvet forks ensure the success of updates because the changes made are not known to legacy nodes, yet they still consider the updated blocks as valid.

2.6 Ethereum

In this section, we briefly present Ethereum [But14], the cryptocurrency with the second highest market capitalisation[6]. Note that this thesis examines Ethereum in the context of when it was still relying on PoW as its underlying consensus mechanism, (i.e., the time period before Ethereum transitioned to a Proof-of-Stake model in September 2022), as well as prior to the EIP-1559 update.

Compared to Bitcoin, Ethereum's block arrival time is not only notably faster (approximately 13 seconds [Eth]), but Ethereum also allows for the creation of so-called *smart contracts*. These are programs which define a set of rules using a Turing-complete programming language, typically Solidity [Dan17], that can be invoked by network participants. An Ethereum account balance is expressed in the underlying currency Ether (ETH) and directly altered via state transitions caused by transactions. Ethereum has two types of accounts, namely Externally Owned Accounts (EOAs), which consist of a public and private key pair, and contract accounts, which do not hold a private key and are therefore unable to sign transactions. The permissionless and composable nature of Ethereum allows anyone to deploy a smart contract that can then be interacted with using transactions. Transactions in the Ethereum Virtual Machine are atomic, meaning that either a transaction succeeds and modifies the global state or it fails and the state remains unchanged.

[6]https://www.coingecko.com/. Accessed: 05-18-2023

2.6.1 Ethereum Virtual Machine (EVM)

The consensus rules governing transaction validity are implemented by the *Ethereum Virtual Machine* (EVM), a low-level stack machine which executes the compiled EVM bytecode of the smart contract. In the EVM, so-called opcodes are the fundamental building blocks of smart contract execution. They are predefined operations that the EVM can perform, such as arithmetic calculations, data manipulation, and control flow operations, allowing the EVM to execute instructions and carry out the logic defined within smart contracts. Operations performed by the EVM consume an amount of *gas*, a virtual unit of account used to measure the computational cost of executing a transaction.

By design, each EVM instruction has a hard-coded[7] gas cost [Woo14]. The total execution cost has to be paid for by the sender of a transaction. Hence, gas fees prioritize and compensate miners. The gas limit sets the maximum gas allowed per block, preventing network overload, while the gas price represents the user's chosen payment per gas unit, incentivising miners to prioritize transactions with higher gas prices.

2.6.2 Gas Mechanism

In Ethereum, the total execution cost for a smart contract consists of two components, namely the gas cost in units and gas price per unit. The gas cost is split into a fixed base cost of 21 000 gas and an execution cost dependent on the instructions executed while running the contract.

Gas Limit. Due to the Turing-completeness of the EVM, the exact computational cost of a transaction cannot be predetermined. Hence, the sender is required to specify a *gas limit*, or the maximum amount of gas that may be consumed. As the computational steps of a transaction are executed, the required gas is subtracted from the paid gas. Once a transaction is completed, any unused gas will be refunded to the sender. Should a transaction try to consume gas in excess of the gas limit, an Out-of-Gas exception is thrown by the EVM. Even though such a

[7]Note that via a hard-fork, the Ethereum Improvement Proposal 150 [But] re-aligned gas costs for instructions involving I/O-heavy operations.

transaction would fail, it would be recorded on-chain and any used gas will not be refunded to the sender. Note that in addition to the per transaction gas limit there is also a *block gas limit*[8], which specifies the total amount of gas that may be consumed by all transactions in a block.

Gas Price. Apart from setting a gas limit, a sender will also have to specify the *gas price*, which refers to the amount of Ether the sender is willing to pay per unit of gas, generally expressed in *wei* (1 wei = 10^{-18}ETH) or *Gwei* (1 Gwei = 10^{-9}ETH). Miners set a cut-off gas price to choose which transactions to include in their memory pool. When constructing a new block, they then choose the transactions with the most lucrative gas prices from their memory pool. A higher gas price will increase the fee which miners receive from a transaction, thereby motivating a miner to include a transaction in a block. The total amount of wei to be paid by a sender is referred to as the *transaction fee* and amounts to the product of the gas price and gas cost.

Gas Price Oracles. The sender of an Ethereum transaction is exposed to the non-trivial task of having to decide on a gas price. Since a higher gas price will increase the likelihood of having a transaction included quickly, there is a clear trade-off between waiting and paying. We define the optimal gas price as the minimum gas price such that the transaction is included in a block within the period of time that the sender of the transaction is prepared to wait for.

In order to avoid risks of overpaying, *gas price oracles* exist [eth20a, Git20b, Git20c, Git20a]. These oracles aim to recommend the gas price a transaction requires in order to be included in a block within a specified time period. Commonly, the recommendation mechanism uses some rule-based approach analyzing the gas prices of previous blocks. We provide a more detailed summary on existing approaches in Section 6.4.

[8] At the time of writing the average block gas limit was around 10,000,000 units of gas.

2.6.3 Uncle Blocks in Ethereum

Unlike Bitcoin, Ethereum has substantially faster block generation intervals. This results in the more frequent occurrence of so-called forks, where multiple blocks are generated around the same time and compete for becoming the head of the chain. To incentivise miners whose blocks did not become part of the main chain to extend the head of the main chain instead of working on their fork, Ethereum introduced the notion of *uncle blocks*. Instead of restricting the ancestors of a block to one, Ethereum follows an *inclusive* approach [LSZ15] where main chain miners can reference uncle blocks. For each such referenced uncle block, additional rewards are distributed to both the main chain miner and the creator of the uncle block. We present a more detailed overview of uncle blocks in Ethereum in Chapter 4.

2.6.4 Applications on Ethereum

By enabling users to build applications that consist of one or more smart contracts, Ethereum essentially allows users to develop DApps, a feature that is not possible in Bitcoin. In Ethereum, the largest type of applications is decentralised finance (DeFi) [WPG+22, ZXE+22] applications, ranging from protocols for loanable funds (e.g., [AAV20, Com19]) to stablecoins (e.g., [Cir20, Mak19]) and decentralised exchanges (e.g., [Uni20, Ego19]). However, the complex logic of these applications tends to rely on rather expensive opcodes in the EVM and therefore tend to consume high amounts of gas, which may become a limiting factor when it comes to adoption of such applications and the frequency to which users interact with them.

2.7 Challenges and Limitations of PoW Blockchains

Proof-of-Work (PoW) blockchains, while being one of the most widely adopted consensus mechanisms, are not without their challenges and limitations. This section explores some of the key obstacles faced by PoW blockchains, including scalability issues, energy consumption concerns, and the potential for centralization. Additionally, it provides an overview of the current research

and developments aimed at addressing these challenges.

Scalability Issues: One of the prominent challenges faced by PoW blockchains is scalability. As the number of transactions and participants on the blockchain network increases, the limited block size and block generation time of PoW can lead to congestion and longer confirmation times. This can hinder the ability of the blockchain to handle a high volume of transactions efficiently. Several solutions have been proposed to mitigate scalability issues, such as layer-two scaling solutions (e.g., Lightning Network [PD16, PZS+16, HJNARP+19] and other layer-two scaling methods [GMSR+19]) and alternative consensus mechanisms (e.g., Proof-of-Stake [GKZ19, KRDO17, BG17] and sharding [LNZ+16, YWY+20, WSNH19]).

Energy Consumption Concerns: Another significant limitation of PoW blockchains is the substantial energy consumption associated with the consensus algorithm. The mining process in PoW blockchains requires significant computational power, which translates into a substantial amount of electricity consumption. For example, in 2023, the Rocky Mountain Institute estimates that Bitcoin consumes approx. 127 terawatt-hours (TWh) a year, suggesting that Bitcoin consumes more electricity annually than all of Norway [Hue23]. This has raised environmental concerns and sustainability issues. Researchers are actively exploring energy-efficient alternatives, such as Proof-of-Stake (PoS), which requires validators to hold and "stake" a certain amount of cryptocurrency to participate in the consensus process, thereby reducing energy consumption.

Potential for Centralization: PoW blockchains are also susceptible to the potential for centralization. In PoW, miners with more computational power have a higher probability of mining new blocks and receiving block rewards. This has led to the emergence of large mining pools and specialized mining hardware [Tay17], creating a concentration of power in the hands of a few entities (e.g., AntPool [Ant23])[9] Centralization in mining can pose security risks and undermine the decentralized nature of the blockchain. Various approaches, including mining pool regulations and consensus protocol modifications, are being explored to mitigate the risks associated with centralization.

[9]For a more detailed breakdown of hashrate distribution for Bitcoin, please refer to `https://www.blockchain.com/explorer/charts/pools`.

Governance and Decision-Making: PoW blockchains often face challenges in making governance decisions and implementing protocol upgrades. Achieving consensus among network participants can be time-consuming and contentious. Disagreements over proposed changes or upgrades can lead to forks and fragmentation within the blockchain community, potentially impacting the network's stability and effectiveness.

User Experience and Adoption: PoW blockchains may present a steep learning curve and technical barriers for average users, limiting their widespread adoption. The complexity of managing wallets, private keys, and the risk of irreversible transactions can hinder user experience and confidence in the system. Enhancements to user-friendly interfaces and education efforts are necessary to facilitate broader adoption.

Research and Developments: To address these challenges and limitations, the research community has been actively working on innovative solutions. One area of research focuses on the development of layer-two scaling solutions, such as the Lightning Network, which allows for off-chain transactions to alleviate congestion on the main blockchain. These solutions aim to improve transaction throughput and reduce fees without compromising decentralization.

Furthermore, alternative consensus mechanisms like Proof-of-Stake (PoS) have gained attention as they offer potential advantages over PoW. PoS relies on validators who hold and "stake" their cryptocurrency as collateral, reducing energy consumption and increasing participation by eliminating the need for resource-intensive mining. Additionally, sharding techniques are being explored to partition the blockchain into smaller, more manageable subsets to improve scalability.

Other innovative approaches include hybrid consensus mechanisms that combine PoW with other consensus algorithms to leverage their respective strengths. For example, projects like Ethereum transitioned from PoW to PoS to address scalability and energy efficiency concerns.

In conclusion, PoW blockchains face challenges and limitations related to scalability, energy consumption, and centralization. However, ongoing research and developments offer promising solutions to overcome these obstacles. Layer-two scaling solutions, alternative consensus mech-

anisms, and hybrid models are among the approaches being explored to enhance the scalability, sustainability, and decentralization of PoW blockchains. By addressing these challenges, the blockchain ecosystem can continue to evolve and meet the demands of a rapidly growing digital economy while ensuring security, efficiency, and inclusivity.

Chapter 3

PoolSim: Simulating Miner Behaviour in Mining Pools

In this chapter, we introduce an open source and discrete event-based mining pool simulation framework called *PoolSim*. The aim is to present a framework which we can subsequently use to better simulate the behaviour of miners under different incentive mechanisms introduced by mining pool reward schemes. The framework can be configured to model the behaviour of miners in one or more mining pools under different network parameters. Furthermore, mining pools can be configured to use different reward distribution schemes, as introduced in Section 2.3.2, allowing for the simulation of different mining strategies that could exist in such scenarios.

The motivation for building *PoolSim* comes from a lack of existing simulation tools with a focus on miner behaviour and different reward schemes. Specifically, we want to be able to simulate more niche reward distribution schemes and reconstruct real-world miner behaviour by defining custom strategies that determine how single or multiple miners behave in a mining pool. In Chapter 4, we demonstrate how *PoolSim* can be used to reconstruct real-world observations of miner behaviour that is detrimental to other miners in a queue-based mining pool. It should be noted that we are not envisioning *PoolSim* to be used beyond simulating mining pools and miner behaviour (e.g., it should not be compared to broader blockchain simulation frameworks that are focused on other aspects, such as consensus and forks).

We first present the design of *PoolSim* and subsequently simulate mining strategies as discussed in existing reseach [ZWW+17]. We then focus on different queue-based mining strategies and extend existing research by simulating scenarios in which multiple miners apply reward-increasing strategies. Furthermore, we provide some insights as to how profitable mining strategies could be in which a miner hops between two queue-based pools.

Contributions

- With the introduction of *PoolSim*, an open-source[1] discrete event simulation framework, researchers on mining pools and reward distribution schemes are no longer required to develop their own simulation software for modeling the behaviour of a miner under a particular mining pool reward scheme.
- In the context of this thesis, we utilise *PoolSim* to simulate different mining strategies in mining pool reward schemes that have not been proven to be fair. Specifically, we will use *PoolSim* with respect to Chapter 4.

3.1 PoolSim: Design and Implementation

Our system is implemented in C++ and is designed to be highly configurable and extensible. The proposed system is primarily composed of a shared library `libpoolsim` which provides the core functionality of the simulator and an executable `poolsim` which takes a configuration file as input, allowing for easily running simulations. In this section, we describe the overall design of the system and provide some of the most relevant implementation details. We give a high-level overview of the design of the library in Figure 3.1.

PoolSim is a discrete-event simulator using a priority queue to store and execute scheduled share events. Share events represent shares generated by a miner, where the time intervals by which shares are submitted are constructed as a randomly generated exponentially distributed

[1] https://github.com/samwerner/PoolSim. Accessed: 2019-03-02

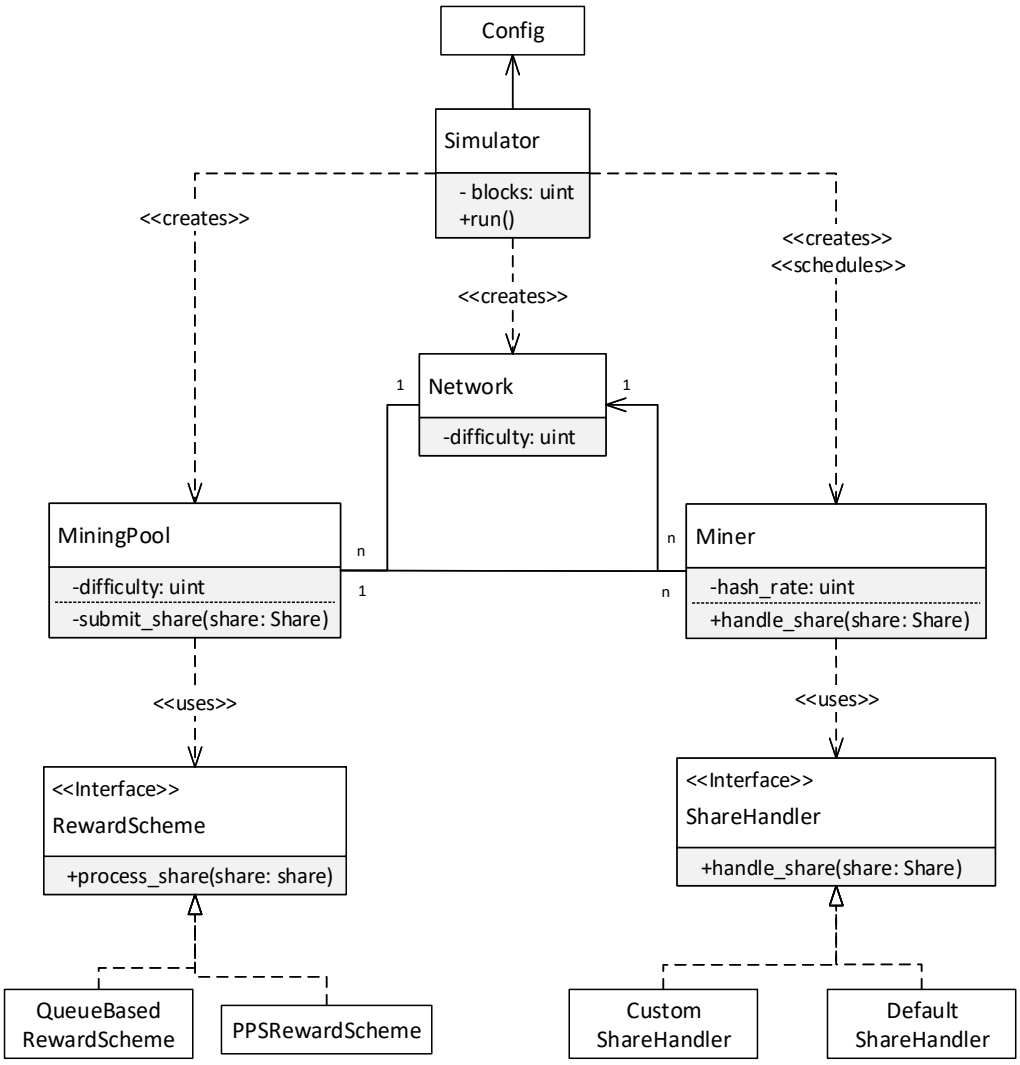

Figure 3.1: Design overview of `libpoolsim`

number with a rate parameter equal to $\lambda = \frac{h}{d}$, where h is the hash rate of a miner and d the share difficulty.

When a share is found, the miner is able to handle the share by using the provided share handler which can be configured or extended. The handler will usually submit the share to the pool it belongs to, but can choose any other behaviour. Upon receiving a share, the mining pool delegates the share to the configurable reward scheme, which implements the logic for distributing rewards to miners. Once all this is done, the next scheduled miner processes the share it found, and this continues until the number of blocks in the simulation is reached. Each reward scheme and miner behaviour can define its own metrics which are serialised with the

Listing 1: PoolSim configuration

```
{
  "blocks": 10000,
  "seed": 120,
  "network_difficulty": 1000,
  "pools": [{
    "uncle_block_prob": 0.01,
    "difficulty": 10,
    "reward_scheme": {"type": "qb",
                      "params": {"pool_fee": 0.05}},
    "miners": {
      "generator": "random",
      "params": {
        "behaviour": {"name": "default"},
        "hashrate": {"distribution": "normal",
                     "params": {"mean": 20, "variance": 5}},
        "stop_condition": {"type": "total_hashrate",
                           "params": {"value": 100}}
      }
    }
  }]
}
```

final results.

Configuration. Many different settings of PoolSim can be configured using a simple JSON configuration file. Rather than describing all the different parameters, we show a minimal sample configuration file in Listing 1.

This configuration runs a simulation for 10 000 blocks with a network difficulty of 1 000. The simulation contains a single queue-based mining pool with a share difficulty of 10 and a probability of its shares becoming uncle blocks of 1%. All the miners in the mining pool have the default behaviour, which is to submit a share to the pool when found. In this example, the miners' hash rates are taken from a configurable normal distribution of mean 20 and variance 5, truncated at 0, as the hash rate cannot be negative. New miners are added to the pool until the total hash rate of the pool reaches 100.

Extending PoolSim. One of the most important features of PoolSim is its extensibility. There are two main parts which are designed to be very easily extended: the mining pool reward scheme and a miner's share handler. In both cases, a base class is provided and custom behaviour can be implemented by subclassing it. In Figure 2, we demonstrate the simplicity

Listing 2: Custom handler which withholds the share with a given probability

```
// maybe_withhold_handler.h
#include <nlohmann/json.hpp>
#include <poolsim/share_handler.h>

class MaybeWithhold : public BaseShareHandler<MaybeWithhold> {
public:
  explicit MaybeWithhold(const nlohmann::json& args);
  void handle_share(const Share& share) override;
private:
  float withhold_prob;
};

// maybe_withhold_handler.cpp
#include "maybe_withhold_handler.h"

MaybeWithhold::MaybeWithhold(const nlohmann::json& args)
  : withhold_prob(args["withhold_prob"]) {}

void MaybeWithhold::handle_share(const Share& share) {
  if (std::rand() >= withhold_prob) {
    get_pool()->submit_share(get_address(), share);
  }
}

REGISTER(ShareHandler, MaybeWithhold, "maybe_withold")
```

of creating new behaviours by presenting a share handler which withholds the share with the probability given in the configuration file.

Once the desired behaviours are created, a new poolsim executable including these can be created by linking against `libpoolsim`. This also allows to integrate *PoolSim* to an existing C++ code base.

3.2 Simulations

In this section, we first use *PoolSim* to reproduce existing research that has used discrete-event simulations to examine profitability of different mining pool strategies. Additionally, we illustrate innovative use-cases of *PoolSim*, which to the best of our knowledge have not been

examined before.

Simulation Setup. For the performed simulations, we assumed static network and share difficulties of 1 000 000 and 10 000, respectively. It should be noted that *PoolSim* allows users to define the logic for having dynamically self-adjusting network and/or share difficulties. The duration of each simulation was set to 100 000 blocks, and a pool size of 1 000 miners has been assumed, unless stated otherwise. An uncle rate of 0% and a pool operator fee of 0%[2] have been assumed. For the attack scenarios in a queue-based pool we have set the condition that if the miner in the position after the attacker in the queue, has a total credit balance equal to 90% or more of the attacker's credits, the attacker executes some deterministic strategy. We define the attacker to be a miner with a total hashrate of 15 GH/s. All simulations are run on an Intel i7-8550U CPU clocked at 1.80GHz, with 16GB of RAM clocked at 1066 MHz. In the current state of our implementation, the simulator is single threaded and we therefore only use one of the 8 threads available on the CPU.

3.2.1 Existing Research

We use *PoolSim* in an attempt to reproduce some of the key contributions of Zamyatin et al. [ZWW+17]. To compare a miner's payouts of mining in a PPLNS scheme to mining in a queue-based scheme, [ZWW+17] construct and evaluate the performance of a two-miner case, in which a small 1 GH/s miner and a large 10 GH/s miner mine under each scheme. Furthermore, the authors identify attack strategies specific to a queue-based pool, which may be employed by a large miner to potentially increase his payout in a two-miner scenario. The following attack strategies are presented by [ZWW+17]:

- **Share withholding:** The attacker withholds any shares if he is at the top of the queue and another miner is within some threshold of his credits. Once the second miner is no longer within the threshold of the attacker then the attacker will continue to submit shares.

[2] As the performed simulations did not involve a PPS reward scheme under which an operator fee would indeed be relevant, we decided to omit this variable.

- **Tactical donation of mining power:** If the attacker is top of the queue and the second miner is within some threshold of his credits, the attacker donates his shares to that miner.
- **Using a second wallet:** If the attacker is top of the queue and the second miner is within some threshold of his credits, the attacker sends his shares to a second wallet that he controls.

We first reconstruct and simulate the normal two-miner case, in which both miners mine honestly absent any attack strategies.

Two-miner Case: Normal. Zamyatin et al. [ZWW+17] find that when comparing the performance between miners in the two-miner case under different reward schemes that the small miner is in fact performing less work per block than the large miner in a queue-based pool. The authors also show that the large miner would have been better off mining in a PPLNS pool (best option).

Miner	Block Ratio			Avg. performed work per block		
	Solo	PPLNS	QB	Solo	PPLNS	QB
Large	1.0	1.000	0.993	1 000 974	1 000 827	1 008 227
Small	1.0	0.999	1.072	1 000 087	1 001 555	932 991

Table 3.1: Blocks mined and rewarded under solo mining compared to mining in a PPLNS and QB two-miner scenario.

In Table 3.1, we show our results of the reconstructed two-miner case. These align with the findings of [ZWW+17], as it can be seen that the large miner performs a notable amount more work than the small miner in the queue-based pool. This is reflected by the ratio of blocks received to blocks mined for the large miner in the queue-based pool, which is worse than had he mined in a PPLNS pool or solo. The relatively high block ratio of the small miner for the queue-based pool is also in line with the findings by [ZWW+17], indicating how the small miner benefits from the large miner absorbing the variance during lucky and unlucky streaks of the pool.

Two-miner Case: Attack. When looking at the attack strategies of share withholding,

Attack strategy	Prop. of avg. credits lost	Miner	Avg. performed work per block	Blocks Rewarded	Blocks Mined	Blocks Ratio
Share withholding	0.188	Attacker	970 208	73 133	71 164	1.028
		Victim	1 295 488	7 005	8 974	0.781
Tactical donation of mining power	0.188	Attacker	972 887	91 192	88 610	1.029
		Victim	1 290 845	8 808	11 390	0.773
Using a second wallet	0.246	Attacker	1 105 026	82 538	82 181	1.004
		Victim	926 686	9 833	9 121	1.078

Table 3.2: Attack simulation results in a two-miner scenario.

tactical donation of mining power, and use of second wallets, the findings by Zamyatin et al. show that the optimal strategy for a large miner is the tactical donation of mining power in a two miner scenario.

We present the results for the two-miner attack scenarios in Table 3.2. As first discovered by [ZWW+17], we also find that the tactical donation of mining power is the most successful strategy. By pursuing this strategy, the attacker can increase his block ratio to 1.029, compared to 0.993 in the normal scenario presented in Table 3.1, and thereby compensate for his initial loss.

Our simulation configuration deviates from the work done by [ZWW+17] when looking at the share withholding strategy. As a miner does not submit the share he withholds, the share is essentially lost, even if this share is a valid solution to the PoW. Therefore, the attacker will find fewer blocks during the same time period as for the other scenarios, in which no work is lost. In [ZWW+17], the length of the share withholding is extended to equal the same duration as all the other scenarios. However, not submitting shares and thus mining fewer blocks poses the risk of being worse off compared to submitting all shares.

Our findings for the effectiveness of the second wallet strategy deviate slightly from the results presented in the original work, as the victim in our simulation performed better than the attacker. However, this difference may be explained by possibly differing implementations of the simulators used. Overall, we were able to reproduce the key findings of [ZWW+17], where

Simulation	Runtime (ms)
PPLNS	5 310
Queue based	4 468
Share witholding	6 528
Tactical donation of mining power	6 805
Using second wallet	7 721

Table 3.3: Execution time of the different simulated two-miner scenarios.

the most effective strategy is the tactical donation of mining power, followed by the withholding of shares and the use of a second wallet, respectively. Given the limited amount of simulations on queue-based mining pool attacks, being able to replicate existing research validates the correct workings of *PoolSim*.

Simulation Performance

In Table 3.3, we present the execution time of the different simulations performed for the two-miner scenario. It is worth noting that using our implementation, the ratio between the network difficulty and the pool difficulty greatly influences the speed of execution, as it changes the number of shares which must be generated before finding a block. However, as this ratio does not affect the behaviour of the attacks we are checking for, we decide to keep it low to speed-up the simulations. In our simulations, this ratio is 100, compared to about 20 000 for most real-world pools.

3.2.2 Normal Multi-miner and Attack Scenarios

In the previous subsection, we have examined how *PoolSim* can be used to reconstruct and review the effectiveness of different queue-based attack scenarios. To receive further insights into the dynamics of queue-based mining pools, we compare the performance of an attacker between mining in different pools containing 1 000 miners each. Table 3.4 shows the *PoolSim* execution time per simulated scenario.

Scenario	Avg. credit lost	Avg. performed work per block	Blocks		
			Rewarded	Mined	Ratio
PPLNS	NA	1,004,706	488.31	488	1.001
QB	0.0020	993,137	494.00	488	1.012
QB with share donation	0.0020	999,205	491.00	485	1.012

Table 3.4: Multi-miner simulation scenarios.

Simulation	Runtime (ms)
PPLNS	331 957
Queue based	158 292
Share donation	166 464

Table 3.5: Execution time of the different simulated multi-miner scenarios.

When comparing the payouts of a miner under a PPLNS scheme to a QB scheme, we find that the attacker did in fact receive a slightly higher number of blocks in the QB pool (494) than in the PPLNS pool (488.31). We note that the attacker received in both pools more blocks than he actually mined. These types of analyses have indeed also already been conducted by [ZWW+17]. However, we additionally simulate a scenario in which the attacker pursues the tactical donation of mining power strategy in a multi-miner pool. We selected the aforementioned strategy as this was the most effective one in the two-miner case. We find that the attacker received fewer blocks (491) than had he mined honestly in the QB pool. The reason why the share donation strategy did not have any noticeable effect in this multi-miner pool is rooted in the existing credit differences between miners. The tactical donation of shares strategy requires certain credit differences to exist, as otherwise giving away shares does not pay off over time. In order to measure the extent of such differences in a given pool, we turn to the average proportion of credits lost per round. For a given round, this measure is expressed as the number of credits of the second miner as a proportion of the total sum of credits of the pool. Looking at this proportion, we find that in the scenario of share donation in the two-miner case, the average proportion of credits lost amounts to 0.188. Interestingly, for the scenario of share donation in a multi-miner case, we find that this figure amounts to 0.002, and is thus significantly lower. In fact, we suggest that the average credit loss is one of the main variables which can be taken

advantage of by an attacker. For example, in a simulation with no attacker, the average credit lost is 0.245. Table 3.2 shows that successful attacks manage to reduce this average.

3.2.3 Queue-based Pool-hopping

This simulation demonstrates *PoolSim*'s capability to simulate condition-based pool-hopping scenarios. As briefly discussed in Section 3.2.1, several analyses on the effects of pool-hopping, as well as on which reward schemes are vulnerable to it, exist. However, we note that there have been no studies on the feasibility of pool-hopping between queue-based mining pools. Hence, we construct and examine a *proof-of-concept* pool-hopping scenario, where a miner submits shares to a pool on the basis of the luck of the pool. Pool luck for a given round r can be defined as

$$l_r = \frac{S_E}{S_A} \cdot 100 \tag{3.1}$$

where S_E is the number of expected shares per round and S_A is the number of actual shares submitted per round.

We construct two mining pools and add a conditional hopping, stating that the attacker will leave the current pool he is in if the number of shares submitted by the pool for the current round is twice the amount as expected, or $l = 50\%$.

We find that the attacker received 251 blocks in total from hopping between both pools, while receiving 249 blocks when mining only in one pool. However, this is based on a rather simple set up, as both pools have log normal hash rate distributions, no attackers and are rather identical. Nonetheless, we have successfully shown that *PoolSim* can be used for simulating simple, as well as more complex conditions.

3.3 Conclusion

Examining the exploitability of potential vulnerabilities embedded within different reward schemes employed by mining pools has evolved into an exciting area of research within PoW cryptocurrencies. In this chapter, we introduced *PoolSim*, a simulation framework targeted for academics or anyone with interest in studying and examining incentive and security mechanisms of mining pool reward schemes. There is ample opportunity for exploration and experimentation using *PoolSim*, especially when it comes to mining pools that utilise less studied reward payout schemes, such as the queue-based approach. Due to the rather limited existing academic research in this area, *PoolSim* provides a valuable tool for simulating and testing new strategies, allowing researchers to delve into uncharted territory and uncover novel insights.

PoolSim allows for a high degree of customisation, where new reward schemes could easily be implemented and tested, or the effectiveness of new attack strategies can be assessed accurately. Additionally, *PoolSim* alleviates the burden of time-consuming and complex implementation tasks for researchers, enabling more efficient exploration of mining pool reward schemes in future research endeavors. We have provided an overview of the design and implementation of *PoolSim*, while also demonstrating the framework's functionality through the reconstruction of relevant academic research. Through these contributions, we hope to foster a deeper understanding of mining pool reward schemes and drive further advancements in this field of study.

Chapter 4

Misaligned Mining Incentives in Queue-based Mining Pools

In this chapter, we focus on queue-based mining pools and show how, due to misaligned incentive structures introduced by the reward payout mechanism, not only individual miners can be made economically worse off, but also that block throughput in the overall network can be negatively affected. We first outline the workings of the uncle block reward distribution mechanism. Next, we then show how an adversary may leverage the uncle reward distribution mechanism of a queue-based mining pool to increase her expected payouts at the cost of other miners in the pool. We then discuss an observed reward-increasing strategy employed by a miner in a queue-based mining pool and reconstruct the observed attack via a discrete-event simulation using *PoolSim* to examine the effect on the mining pool. Lastly, we propose a modification to the studied uncle reward distribution policy, which would obviate reward-increasing strategies rooted in the former.

Contributions

- We show how for an Ethereum queue-based mining pool reward scheme, the uncle block distribution mechanism can be exploited by an adversarial miner to increase their earnings by employing an attack we call "uncle traps".

- We demonstrate how uncle traps can be employed by an adversarial miner to harm the overall network by increasing the uncle rate, i.e., lowering the block throughput. We use *PoolSim* to construct discrete event simulations of the proposed attack.
- In order to better understand the feasibility of the proposed attack, we analyse empirical mining pool data to show that this form of attack has indeed been successfully executed in the past in a queue-based mining pool on Ethereum.
- We propose a modification to the uncle block distribution mechanism, which would remove the attack vector for uncle traps.

4.1 Background

In this section, we first explain the uncle block mechanism that was a core component of Ethereum's PoW mechanism until Ethereum switched from PoW to Proof-of-Stake in September 2022. Note that at the time when this research was first conducted, Ethereum was still using PoW. We then examine existing research on manipulation of mining pool reward schemes.

4.1.1 Block Rewards in Ethereum

In Ethereum, at the time this research was conducted, each full block was rewarded with a static reward of approx. 2.2 ETH[1], and any fees paid by users for transactions included in the block. Block arrival times follow a Poisson distribution with the rate parameter $\lambda = \frac{H}{D}$, where H is the network's total hash rate and D the network difficulty of the PoW. In Bitcoin, the PoW difficulty is adjusted every two weeks to maintain a target block generation interval of 10 minutes. In Ethereum the difficulty is adjusted dynamically after every block while the target block interval amounts to approximately only 15 seconds [But14].

Multiple PoW solutions found roughly around the same time can create two (or more) parallel competing branches in the underlying blockchain due to network latency [DW13, CDE+16].

[1] ETH is the underlying unit of exchange in Ethereum

Eventually, the branch supported by the majority of the computational power in the network becomes the main chain, i.e. the sequence of blocks accumulating the most PoW effort since the genesis block. All other branches are discarded and receive no rewards. To achieve a faster convergence to a single chain, Ethereum leverages a reward scheme for forked blocks, similar to the notion of *inclusive blockchain protocols* [LSZ15]. Miners of main chain blocks can reference forked blocks, which are then referred to as uncle blocks. Each referenced uncle block results in an additional reward being distributed to both the main chain miner and its creator. Participants mining on conflicting branches are hence incentivised to rejoin the main chain, as they are guaranteed reimbursement for otherwise wasted computational effort.

Each block included in the main chain can reference up to two uncle blocks and receive a small reward of $\frac{1}{32}$ of a full block reward per referenced uncle. However, the reward paid to the miner of each referenced uncle block varies. The reward diminishes depending on how distant of an ancestor the uncle block is relative to the main chain block it was referenced by. An uncle block is rewarded for being up to six generations away from the included block. Hence, the reward is computed as

$$U = (U_n + 8 - B_n) \cdot \frac{B}{8}, \tag{4.1}$$

where U is the uncle block reward, U_n the uncle block number, B_n the number of the block included in the main chain, and B the reward for a full block [com18].

4.1.2 Mining Pool Attacks

While mining pools are intended to benefit pool participants through a more steady payout stream, they are subject to different types of attacks. Previous research [LSP+15, CB14] has shown that *block withholding* attacks, whereby a malicious miner withholds blocks from the pool, yet still receives a reward from the pool operator for her overall performed work, can cause mining pools to suffer tremendous financial losses. Vasek, Thornton and Moore [VTM14] have shown how mining pools are negatively affected by Distributed Denial of Service (DDoS) attacks. Laszka, Johnson and Grossklags [LJG15] have examined game-theoretic aspects of

attacks between mining pools.

Primarily, adversaries will try to increase their own expected payouts at the cost of other miners by exploiting vulnerabilities of the underlying reward scheme. With regards to the queue-based payout scheme, different attack strategies aimed at exploiting the non-uniform credit reset mechanism have been identified. Zamyatin et al. [ZWW+17] propose attack scenarios following real-world observations in Ethpool, whereby an adversary may strategically donate her computational power to other miners in the pool in order to manipulate the queue constellation with the aim of receiving larger credit differences. These attacks were further studied by Holland et al. [HCD+18]. However, the aforementioned attacks, both explicitly focus on the non-uniform credit reset mechanism and ignore any potential vulnerabilities stemming from the uncle reward distribution policy of the pool.

4.2 Attack Description, Observation and Reconstruction

We propose a Sybil attack that leverages the reward-increasing opportunities under a scheme of random distribution of uncle block rewards. According to this scheme, uncle block rewards are distributed randomly among miners in the pool without accounting for differences in the work performed. The selected miner receives the full uncle block reward. For instance, a miner with 2 GH/s has the same chance of receiving a found uncle block as a miner with 10 MH/s.

The attack entails the division of hashing power between a set of smaller miners and the deliberate increase of the pool uncle rate. We introduce the notion of *uncle traps*. These describe miners with conspicuously small hash rates that serve the sole purpose of increasing the likelihood of receiving uncle block rewards in the aforementioned scheme.

We define a fair pool, where fairness refers to the expected reward of an individual miner being proportional to the shares submitted to the pool by that miner in relation to the total shares submitted to the pool. We use the notion of a *round* as the time interval between two blocks found by the pool – these may be uncle blocks or full blocks.

4.2.1 Attack Modelling

Definitions

Define D as the network difficulty and B and U as the block and uncle block rewards, respectively. The fairness assumption means that a miner i with hash rate h, mining in a pool of N miners, where the total hash rate of the participating miners is

$$H = \sum_{i=1}^{N} h_i \quad , i = 1, \ldots, N \,,$$

and the expected duration of a round t_R is equal to

$$E[t_R] = \frac{D}{H} \,,$$

should have an expected reward per round of

$$E[R] = \frac{h \cdot E[t_R]}{D} \cdot E[R_P] = \frac{h}{H} \cdot E[R_P] \,, \tag{4.2}$$

where $E[R_P]$ is the expected reward of the pool.

Attack Model

The total number of miners in the pool is defined as N, consisting of the miner i and all other miners, denoted N_O. Each miner can earn a reward by either being at the top of the queue when a block is found, i.e. having the highest number of accumulated credits, or by receiving an uncle reward. Define p as the network probability of finding an uncle block in each round and $(1-p)$ as the probability of finding a regular block. From the pool fairness (4.2), we know that the individual miner i will receive an expected reward from full blocks of

$$E[R_B] = \frac{h_i}{H} \cdot (1-p) \cdot B \,. \tag{4.3}$$

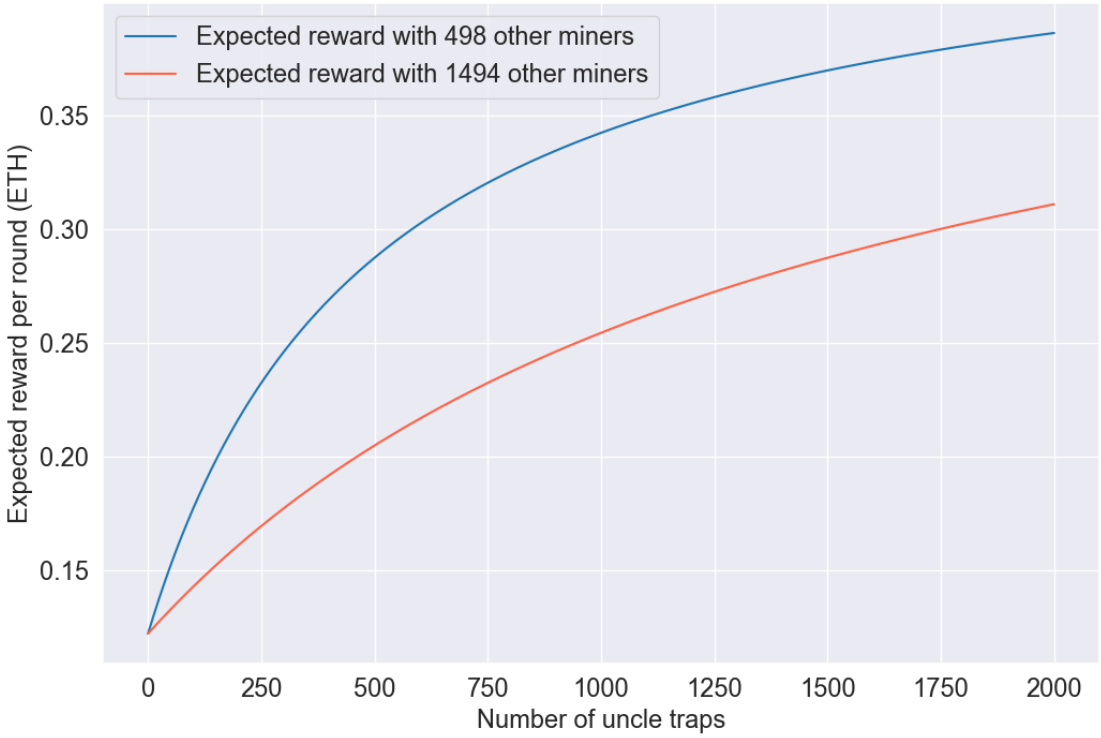

Figure 4.1: The attacker's expected reward per round as a function of the number of uncle traps employed.

Additionally, uncle blocks are distributed randomly between the miners, adding an expected reward from uncle blocks of

$$E[R_U] = \frac{1}{N_O + 1} \cdot p \cdot U \,. \tag{4.4}$$

We now consider a scenario where a miner in the pool, defined as the attacker has a hash rate of h_A. The attacker controls a number of miners, denoted N_A, where N_A refers to all miners controlled by the attacker, including uncle traps. The expected reward function from full blocks remains unaltered for the attacker. However, the expected uncle block reward changes due to the introduction of uncle traps. The expected reward from uncle blocks is now

$$E[R_U] = \frac{N_A}{N_O + N_A} \cdot p \cdot U \,, \tag{4.5}$$

per round. Numerical examples of Equation 4.5 can be found in Figure 4.1.

Combining the two possible sources of reward from (4.3) and (4.5), we find the attacker's total

expected reward as

$$E[R_A] = \frac{h_A}{H} \cdot (1-p) \cdot B + \frac{N_A}{N_O + N_A} \cdot p \cdot U \,. \tag{4.6}$$

Hence, an attacker could substantially increase her expected reward, by dividing her mining power between a large number of smaller miners, thereby creating uncle traps. In Ethereum, additional addresses can be created at no financial cost, making this strategy feasible in practice. However, in order to be recognised by the pool operator, a minimum hashing power has to be maintained, imposing a lower limit on the divisibility of computational power.

In addition to splitting hashing power, an attacker can increase her reward by intentionally increasing the pool's uncle rate. This can be achieved by withholding a full block from the pool operator until some other miner has found a block and only then submitting it. The probability of event F that the attacker finds a block in any given round is

$$P(F) = \frac{h_A}{H} \,. \tag{4.7}$$

For a proportion of some long period of time, the attacker will be at the top of the queue. This proportion $P_{\,TOP}$ is found to be

$$P_{\,TOP} = \frac{E[R_B]}{B} \,. \tag{4.8}$$

Conversely, the proportion of time the attacker is not at the top of the queue is $P_{\,\overline{TOP}} = (1 - P_{\,TOP})$. To pursue the outlined strategy, the attacker will withhold blocks, unless she is at the top of the queue. Hence, she will only force an uncle block, if she finds a block and is not top of the queue. The number of blocks I the attacker finds over time period T, consisting of T/t rounds of duration t, for which she is not top of the queue is equal to the expected maximum number of intentional uncle blocks. We find the expected value of I to be

$$E[I] = (1 - P_{\,TOP}) \cdot P(F) \cdot \frac{T}{t} \,.$$

From Equations (4.7) and (4.8) we find the expected number of intentional uncle blocks as

$$E[I] = \frac{h_A}{H} - \frac{{h_A}^2}{H^2} \cdot (1-p) \cdot \frac{T}{t}\,.$$

Therefore an adversary has two levers to exploit the random uncle block reward distribution scheme. Firstly, she may split her hashing power between multiple miners to increase the likelihood of receiving an uncle block. Secondly, she can intentionally increase the pool uncle rate, increasing the total number of uncle blocks she can attempt to gather.

4.2.2 Attack Execution

We now proceed to reconstruct the observed attack step by step.

1. An attacker generates N_A Ethereum addresses, where $N_A - 1$ will be the number of uncle traps

2. The attacker iterates over the generated uncle trap addresses and computes s shares per address. We assume $s = 1$ for simplicity, although the actual number may be higher in reality.

3. Each time a generated share is the valid solution to the network's PoW puzzle:

    - If one of the attacker's miners is at the top of the queue, publish block to pool instantly.
    - Else, withhold the block until another block is found in the network, then publish to the pool, generating an uncle.

Additionally, the attacker maintains one larger miner in the pool. For that miner, she follows the behaviour outlined in (3) above. Hence, she will force uncle blocks if none of her miners are at the top of the queue and immediately publish the block otherwise.

4.2.3 Real World Observations

Historic observations about the state of Ethpool suggest that there has been at least one occasion where a large miner did pursue the aforementioned reward-increasing behaviour over a longer period of time [Bit18]. It was observed that a swarm of 7 500 miners with conspicuously small hash rates, reportedly ranging from 2 MH[2] per second to 6 MH/s, were mining in the pool, all of which were orchestrated by the same adversary[3]. We shall refer to this pseudonymous beneficiary account as the attacker. Even though it is not possible to retrieve historic hash rate information for Ethpool participants, it was possible to verify that the aforementioned adversary was a major beneficiary of the rewards collected by the observed family of smaller miners during the period of mid-May to mid-July 2018 by tracing payments on the Ethereum blockchain. We extracted the addresses of all recipients of uncle and full block rewards from Ethpool for the two month period and checked if any of these addresses made a transaction to the attacker. Over the course of this period, we found that the attacker managed to receive 19.14% of the total uncle blocks found by Ethpool spread across a set of 148 unique miners participating in the pool. Figure 4.2 displays the total number of uncle blocks found per day by Ethpool over the stated time period and compares it to the share of uncle blocks received by the attacker. The attacker's share of the total received uncle block rewards is substantially higher compared to her 5.55% share of the total full blocks mined by the pool throughout the same time period.

For the examined time period, the average total hash rate of the Ethereum network was 276.16 TH/s[4]. During that time, Ethpool found 1.54% of the total number of blocks found by the network and thus Ethpool's overall hash rate at that time was approximately 4.24 TH/s. Hence, from the attacker's share of the total number of full blocks mined by Ethpool, we know that the attacker also accounted for approximately 5.55% of the pool's total hash rate, or 185.74 GH/s[5]. A total of 7 500 miners with an average hash rate of approximately 4 MH/s per miner, operated by the attacker, would require 30 additional GH/s, invested in uncle traps.

[2] 1 Megahash = 10^6 hashes
[3] miner address: 0x7dE44b1F1527486a16FF586eF301B6b62dA6aC11
[4] 1 Terahash = 10^{12} hashes
[5] 1 Gigahash = 10^9 hashes

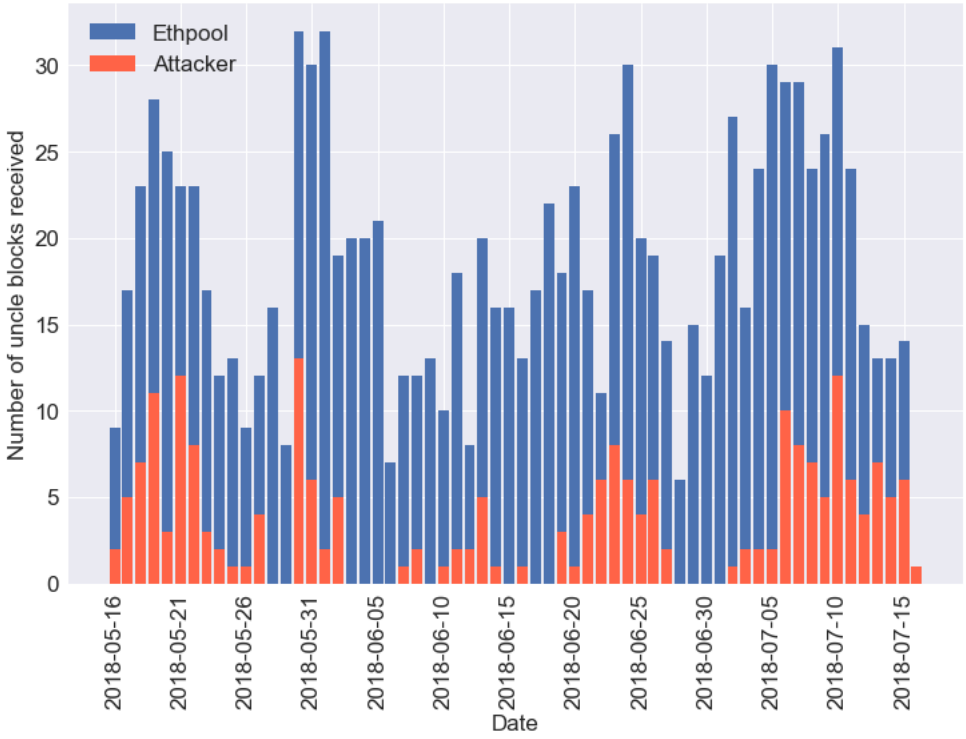

Figure 4.2: The uncle blocks the attacker received between mid May and mid July 2018 in Ethpool compared to all uncle blocks found by the pool during that period.

These derivations are in line with the observations concerning uncle blocks shown in Figure 4.2, which presumably reflect the performance of the uncle traps harvesting the uncle block rewards. An overview of the total reward distribution of the attacker is shown in Figure 4.3.

A further interesting observation lies in the uncle rate of Ethpool compared to the network's uncle rate for the examined time period, as shown in Figure 4.4. With an average uncle rate of 22.92%, the pool lies slightly above the network's rate of 19.52%. Ethpool's uncle rate standard deviation of 0.1090 notably exceeds the network's uncle rate standard deviation of 0.0245. This is explained by the deliberate increase of the uncle rate by the attacker, as described above.

4.3 Proposed Mitigation

Apart from the general challenge of random number generation in a publicly verifiable manner, the main problem of the studied uncle reward policy lies in not accounting for hash rate differences between miners in a pool. Distributing uncle blocks following the same single queue-based

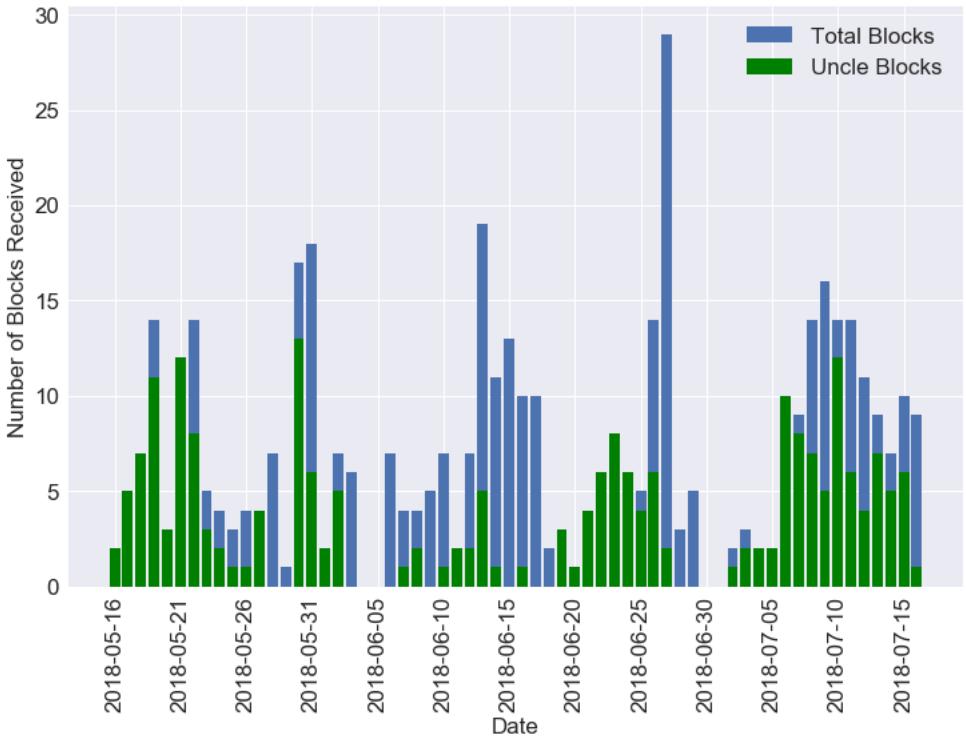

Figure 4.3: The number of uncle blocks the attacker received per day relative to the total number of blocks she received per day in Ethpool between mid May to mid July 2018.

scheme as for full blocks would lower the expected reward of mining in the pool below that of solo mining and does thus not serve as a mitigation. Our proposed solution to the problem entails a hash rate-weighted random allocation of uncle blocks to participating miners. Let the probability of receiving an uncle block reward be $\frac{h_i}{H}$ for each miner i, where h_i is the hash rate of that miner and H is the total hash rate of the pool. As in a PPLNS scheme, miner hash rates h_i are computed as $h_i = \frac{s_i}{S}$, where S is a large number – e.g. a multiple of the network difficulty – and s_i are the shares submitted by a miner i over the period of S shares. Further, we define h_A as the total hash rate of the attacker. The attacker's expected reward per round from receiving uncle blocks, when dividing her hashing power between N_A miners will then be

$$E[R_U] = \sum_{i=1}^{N_A} \frac{h_i}{H} \cdot p \cdot U \,. \tag{4.9}$$

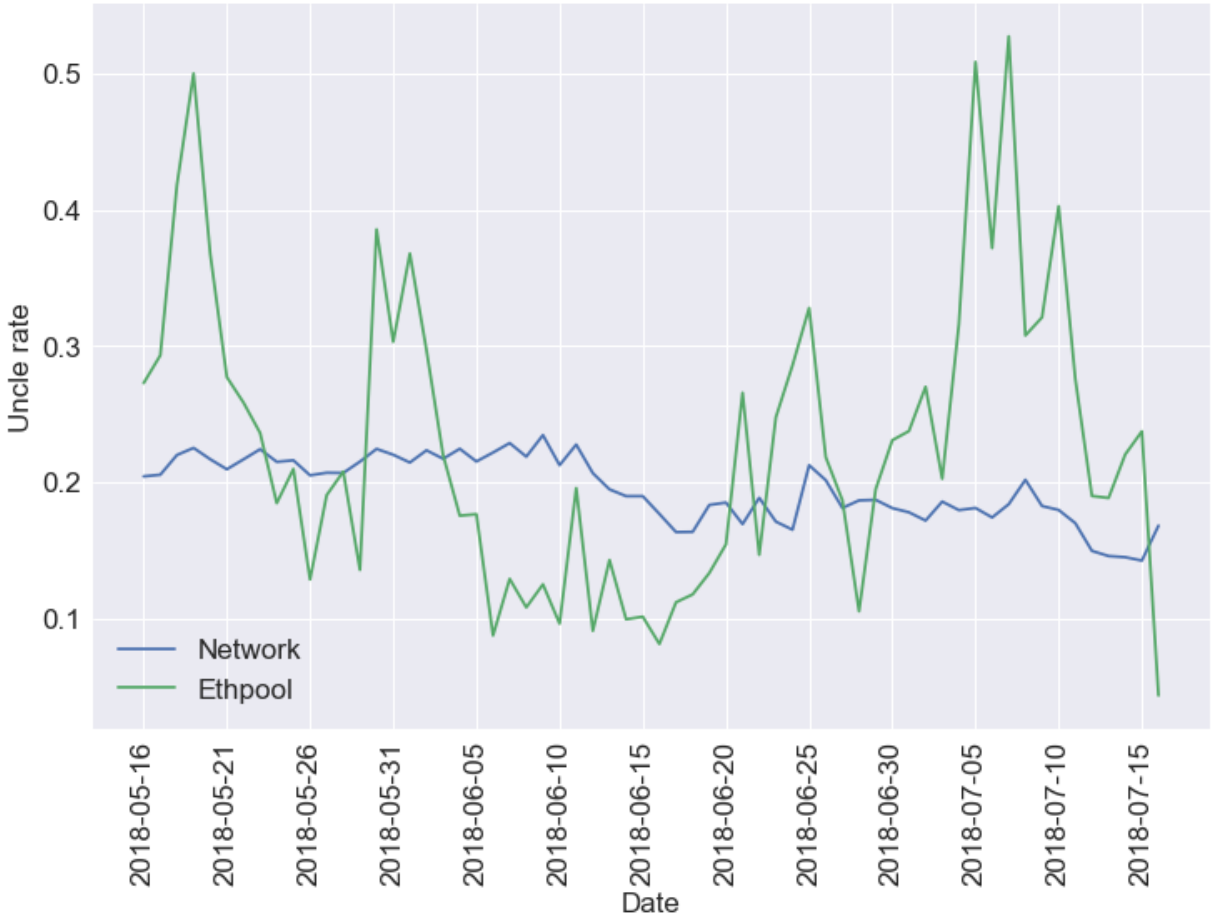

Figure 4.4: The uncle rate of the Ethereum network compared to the uncle rate of Ethpool between mid May and mid July 2018.

From Equations (4.9) and (4.3), we find the attacker's total expected reward as

$$\begin{aligned} E[R_A] &= \frac{h_A}{H} \cdot (1-p) \cdot B + \frac{h_A}{H} \cdot p \cdot U \\ &= \frac{h_A}{H}((1-p) \cdot B + p \cdot U) \end{aligned}$$

By introducing a hash rate-weighted allocation of uncle block rewards, we eliminate the number of miners N_A – operated by the attacker – from the expected reward function. Hence, an attacker is no longer able to increase her profit function from dividing hashing power between many miners and fair uncle block distribution is assured. It should be noted that this mitigation does not preclude the deliberate increase of a pool's uncle rate.

4.4 Discrete-event Simulation of Uncle Traps

In this section we use a discrete-event simulation[6] to examine the effectiveness of the reconstructed attack strategy. We compare the performance of an attacker between different simulated mining pool scenarios.

4.4.1 Simulation Setup

We simulate a queue-based mining pool for a duration of 200 000 blocks using a discrete-event simulator developed in C++. All simulations were executed on an Ubuntu server with an AMD EPYC 7401P processor. For the constant uncle rate, the average uncle rate of the network during the observed attack in Ethpool, namely 0.195235, is taken. Additionally, we assume static network and share difficulties of 200 trillion and 3.6 billion, respectively. The total computational power of Ethpool during the observed attack was equal to approx. 4.24 TH/s, yet the exact hash rate distribution remains unknown. However, Zamyatin et al. [ZWW+17] show that the hash rates in Ethpool resemble a log-normal distribution. Hence, we construct a mining pool by sampling from a log normal distribution for the hash rates of the pool participants until the overall pool size is equal to approx. 4.24 TH/s, including the prespecified hash rate of the attacker. The distribution of sampled miners can be seen in Figure 4.5.

We examine three specific scenarios. The first scenario is a reference scenario referred to as *Honest*, which represents regular mining under a queue-based payout scheme, absent any attack strategy. In this reference scenario, the attacker employs her full computational power of 215.73 GH/s for a single account. For the second scenario, called *Attack*, we simulate the reconstructed attack. In this scenario, the attacker deploys 185.73 GH/s from one account and intentionally increases the uncle rate as outlined in Section 4.2. Additionally, the attacker spreads a total of 30 GH/s equally across 7 500 uncle traps she controls. Lastly, we simulate the same attack behaviour under the hash rate-weighted uncle block reward distribution policy we proposed in the previous section. We refer to the third simulated mining scenario as the

[6]This is an early version of *PoolSim*, which was then later revised and published in [WP20].

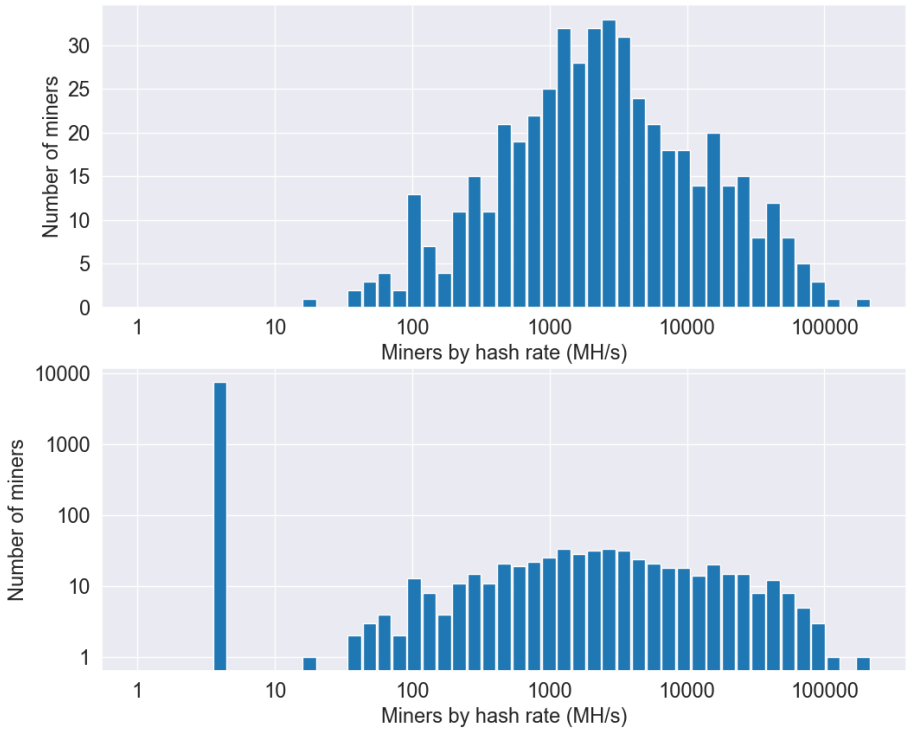

Figure 4.5: The miner distribution in a mining pool with no uncle traps (top) opposed to with uncle traps (bottom).

Mitigation case.

4.4.2 Simulation Results

To evaluate the performance of the attacker in the three distinct mining pool scenarios, we focus on the number of full and uncle blocks mined and rewarded. Additionally we assess the success of the attacker by examining the reward per invested MH in ETH. We can compute the former as

$$\textit{Reward per invested MH} = \frac{(B \cdot b_r) + (U \cdot u_r)}{(TS \cdot d)/1\,000\,000}, \tag{4.10}$$

where for a given miner m_i, TS is the total number of shares submitted, b_r the total number of full blocks received by a miner, and u_r the total number of uncle blocks received by a miner.

Honest Scenario

For the *Honest* scenario, Table 4.1 shows that the attacker (behaving honestly) was rewarded for a substantially lower number of uncle blocks relative to the number of uncles she mined. Assuming the average uncle block reward was 1.6875 ETH[7], this would amount to a total loss of 3 199.5 ETH, or 684 213.01 USD[8] for the attacker compared to mining solo. Based on the uncle rate of 19.52%, the block duration of 200 000 blocks, as well as on the constructed pool consisting of 498 miners, the expected number of uncles rewarded per miner regardless of hash rate is 76.31. This shows how large miners with above average hash rates in the pool are inherently at a disadvantage, compared to mining solo, due to the uncle block reward distribution.

Attack Scenario

In the *Attack* scenario, the attacker mined a significantly lower number of full blocks, a logical consequence of the uncle rate spiking. The uncle traps have resulted in 43 865 uncle blocks

[7] the average of the possible uncle block rewards

[8] with an ETH price of 213.85 USD. www.coinmarketcap.com. Accessed: 10-03-2018

		Scenarios		
		Honest	Attack	Mitigation
Full Blocks	*Mined*	8 126	329	329
	Rewarded	8 065	6 651	6 651
	Ratio	0.9925	20.2158	20.2158
Uncle Blocks	*Mined*	1 968	9 785	9 785
	Rewarded	72	43 865	2 353
	Ratio	0.0366	4.4829	0.2405
Reward per invested MH (ETH$\times 10^{-8}$)		1.2075	4.6812	1.1921

Table 4.1: Simulation results for a honest mining pool scenario and an attack scenario with uncle traps.

being rewarded to the attacker, an increase by a factor of 609.24 relative to the number of received uncle blocks in the *Honest* scenario. The attacker was rewarded 34 080 uncle blocks more than she actually mined. When assessing the overall performance of the attacker, it can be seen that the attacker was able to increase her reward per invested MH by a factor of 3.88 from 1.2075×10^{-8} ETH to 4.6812×10^{-8} ETH overall.

Mitigation Scenario

For the *Mitigation* case, the attacker received only 5.36% of the uncle block rewards she received in the *Attack* scenario. This is in line with our formal evaluation of the reward scheme in Section 4.3. The number of uncles received is still substantially higher than for the *Honest* scenario. However, the attacker was rewarded 17.53% fewer full blocks, due to the continued use of the uncle rate spiking strategy. Overall, the reward per invested MH is the lowest for the Mitigation case, being slightly lower than for the *Honest* scenario.

4.4.3 Other Interesting Observations

Given that the attack strategy under the current uncle block reward distribution policy proved to be very profitable, we investigate its effect on other pool participants. The top plot of Figure 4.6 displays the reward distribution for full and uncle blocks among miners of different hash rates over the course of 200 000 blocks, absent any attack strategy. The bottom plot shows the block distribution for the uncle trap attack scenario. When comparing these two plots, one can see in the attack scenario (bottom) that the majority of uncle block rewards are being absorbed by miners of very small size (red line), the uncle traps. The high drop in uncle reward density for medium and large sized miners can be explained by the altered hash rate distribution for the pool under the presence of uncle traps (see Figure 4.5), which absorbed 43 860 uncle block rewards.

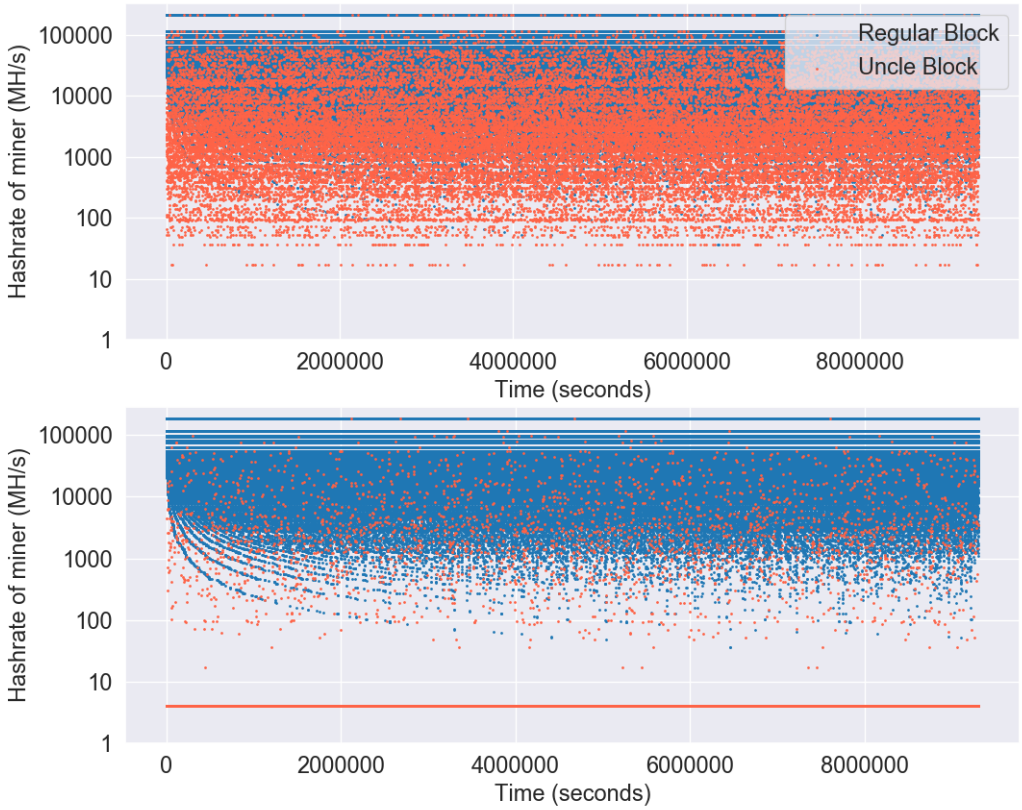

Figure 4.6: Small miners (red line in bottom plot) absorbing the uncle block rewards in the *Attack* scenario, compared to the *Honest* case (top).

The effect of uncle traps on the performance of all pool participants can also be seen by comparing the performance of miners based on their computational efforts, as shown in Figure 4.7. As the current uncle reward distribution mechanism does not account for differences in hash rates, miners of different hash rates are affected equally negatively. This is different from the previously studied attacks described in Section 4.1.2. Hence, an adversary orchestrating an uncle trap attack can not only increase her own reward, but also harms the pool participants as a collective.

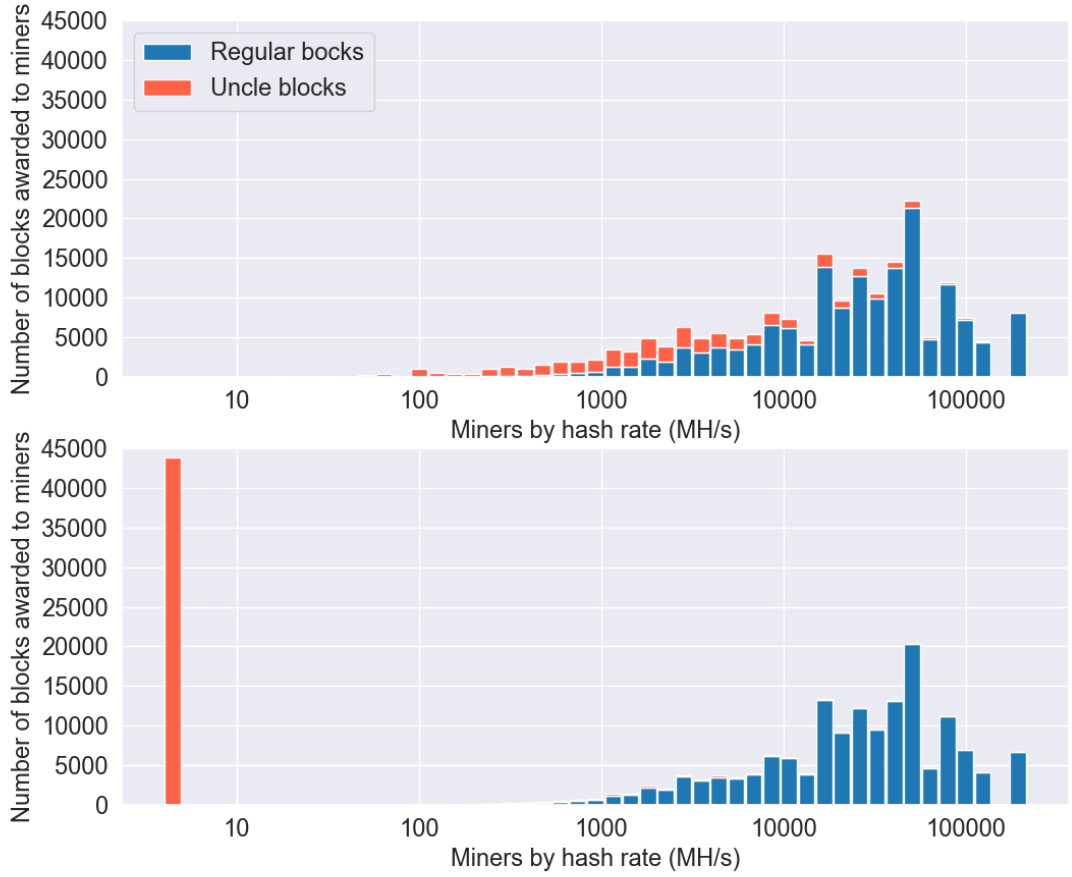

Figure 4.7: Number of full and uncle blocks awarded to miners based on hashrate (MH/s) on a logarithmic scale in a pool with no uncle traps (top) and with uncle traps (bottom) for 200 000 blocks.

4.5 Follow-up work

Since the original research of this chapter was published, there has been some follow-up work that we would like to highlight.

The authors of [DZ21] investigate the impact of selfish mining (valid blocks are withheld from the network in order to gain some sort of competitive advantage over the network), on blockchain systems, specifically focusing on the risks posed by hiding and strategically broadcasting mined blocks. In the context of Ethereum, the authors examine the effects of selfish mining on uncle blocks. They note that the existing uncle incentive mechanism in Ethereum exacerbates the problem by reducing the cost of selfish mining failure and lowering the profitability threshold for selfish mining pools. In response, the authors propose feasible modifications to the uncle incentive mechanism to counteract selfish mining in Ethereum. They analyse the behaviours of miners in relation to uncle blocks when selfish mining occurs by constructing models. Additionally, they introduce a practical uncle incentive mechanism that does not require a strictly ordered strategy for handling generations of uncle blocks.

On the 18th of October 2021, the Ethpool solo-mining pool that was analysed as part of the research presented in this chapter has ceased its operations and completely closed down. To the best of our knowledge, the queue-based reward payout scheme is no longer being utilised by any mining pool. However, the scheme itself is still very interesting due to its simplicity and could easily be reintroduced in the context of PoW chains, such as Bitcoin. A reintroduction of the scheme should first require a formal approach to modelling the fairness of the scheme and potentially making changes to ensure fairness to all participants.

4.6 Conclusion

Building on the anomalies observed in Ethpool data, our evaluation focused on a Sybil attack strategy specifically designed to exploit the random uncle block reward distribution system within a queue-based mining pool. Through a meticulous formal reconstruction of the observed

attack, we successfully identified two key levers that an attacker can manipulate to increase their reward. To validate the effectiveness of this attack strategy, we conducted a comprehensive discrete-event simulation, which further substantiated our findings.

Our research also sheds light on the significant negative impact that an uncle trap attack can have on mining pool participants. However, we propose a potential solution to mitigate these effects by considering the hash rate differences among miners in the uncle block reward policy. By implementing an altered distribution mechanism, where the likelihood of a miner receiving the full uncle reward is directly proportional to their hash rate, we can foster a more equitable and fair pool environment. Such a scheme would not only ensure pool fairness but also render the outlined Sybil attack strategy ineffective.

Through our work, we highlight the importance of addressing vulnerabilities in mining pool reward systems and propose practical measures to enhance security and fairness. By considering the hash rate of miners in the distribution of rewards, we can promote a more balanced and robust mining ecosystem. These recommendations aim to fortify the integrity of mining pool operations and protect participants from potential exploitative attacks.

Chapter 5

(Un)Stable Block Throughput: Feedback Loops and Difficulty Adjustment Algorithms

To ensure stable transaction throughput in PoW blockchains, the difficulty of the PoW problem is adjusted in response to changes in the miners' computational power by a difficulty algorithm (DA). However, without careful design the DA can expose vulnerabilities, which when exploited by miners, lead to inappropriate difficulty levels and thus patterns of instability in the transaction throughput. In general, this issue arises in blockchains that lack a consistent amount of computational power due to some miners directing their resources towards other blockchains especially as profitability varies. For instance, such patterns have been observed even in Bitcoin Cash [Res19, Bit17] (BCH), the cryptocurrency with the 4^{th} highest market capitalisation at one point[1].

In this chapter, we formally model a DA designed to stabilise transaction throughput even in chains without consistent computational power. To this end, we propose adjusting the difficulty after every block using exponential smoothing, a popular approach for time series data. To justify the specifics of the proposed algorithm we provide a case study on the DA used by

[1]Data obtained from: `https://coinmarketcap.com`. Accessed: 26-03-2020.

BCH up until November 15, 2020, (namely `cw-144`) and investigate inherent vulnerabilities which caused misaligned incentives for network participants. We discover that even economically rational (i.e. profit-seeking) miner behaviour leads to severe instabilities in transaction throughput due to a positive feedback loop in block solve times resulting from the design of `cw-144`. We present desirable properties of a proposed DA called NEFDA (Negative Exponential Filter Difficulty Algorithm) and show how they remove the cyclical positive feedback mechanism. Furthermore, we demonstrate through simulations how NEFDA performs under different scenarios and compare it to BCH's `cw-144`. We find that the proposed DA would be an improvement over BCH's `cw-144` DA and suggest that it could be applicable to any PoW blockchain when configured appropriately.

Contributions

This chapter makes the following contributions:

- We conduct an empirical analysis of the `cw-144` BCH DA and examine how the DA's design encourages coin-hopping behaviour, which then leads to the formation of a positive feedback loop in block solve times.
- We quantify the impact this positive feedback loop has on the transaction throughput by measuring the distribution of blocks in one-hour periods.
- We examine the extent to which miners adopt coin-hopping strategies in response to changes in BCH's profitability and derive a DA (NEFDA) which discourages coin-hopping strategies and present its additional properties that limit high variations in block solve times.
- Through simulations, we verify our claims and study the impact of various configurable parameters, arguing that the proposed DA can be customised to meet the requirements of other blockchains.

- We define desirable properties exhibited by NEFDA and prove their benefits by comparing the performance of NEFDA with `cw-144` in a simulated mining scenario.

- Two proposals [fTCL20, Sec20] have been put forward to replace BCH's DA known as `cw-144`. The DA we present is similar to these proposals and is explicitly referenced[2] by one of them [fTCL20], which was implemented as the new BCH DA on November 15, 2020.

5.1 Bitcoin and Bitcoin Cash

In this section, we introduce readers to the core workings of Bitcoin (BTC) [Nak08] and Bitcoin Cash with particular focus on timestamps and difficulty algorithms. As BCH is a hard fork of BTC, the theoretical foundation and even practical implementation of both is mostly similar. Note that when we refer to BCH, we are referring to the Bitcoin ABC [Bit20] full node implementation. Unless otherwise stated, the reader can assume any mention of Bitcoin applies to both BTC and BCH.

5.1.1 Bitcoin and Mining

In Bitcoin, the target is a 256-bit number encoded in the `nBits` field of the block header and it has a maximum value of `0x1d00ffff` ($\approx 2^{224}$). The notion of *difficulty* expresses the ratio of the maximum target to the current target, $D \approx \frac{2^{224}}{target}$. As `SHA-256` computations produce random yet deterministic outputs, each attempt can be modeled as a Bernoulli trial with success probability $\frac{target}{2^{256}}$. Therefore, the expected number of hashes that need to be computed to mine a block at a specific difficulty D is approx. $D \cdot 2^{32}$. As attempts are independent of one another, the time it takes to mine a block, namely the block solve time, follows an exponential distribution with a rate parameter $\lambda = \frac{H}{D \cdot 2^{32}}$, where H is the total hash rate of the network. The expected solve time is then, $\frac{1}{\lambda} = \frac{D \cdot 2^{32}}{H}$. In Bitcoin, the desired block solve time is 10 minutes,

[2]The reference has been made to an earlier pre-printed version of [IWSK21]

which is maintained by adjusting the target depending on the current hash rate estimate; the lower the target, the more difficult it becomes to find a PoW solution.

The block reward started at 50 Bitcoins and halves every 210 000 blocks, or approx. every 4 years (e.g. currently it is set at 6.25 Bitcoins).

5.1.2 Block Timestamps

Each block header contains a UNIX timestamp indicating when the block was mined. As clock synchronisation is a well-known problem in distributed networks, block timestamps may not necessarily be in monotonically increasing order. To ensure the blockchain time advances, the Bitcoin protocol requires blocks to have a timestamp greater than the median timestamp of the previous 11 blocks, also known as the *Median Time Past* (MTP). Additionally, nodes also enact a convention by which they accept new blocks only if their timestamp does not exceed the network adjusted time[3] by more than 2 hours. For a more in-depth analysis of timestamps, potential attacks and improvements we direct the reader to [Sza18, Bov11, zaw18].

5.1.3 Miner Incentives

Blockchains which use the same PoW puzzle can be seen as being PoW compatible from a miner's perspective, i.e. miners can switch between them with no additional hardware overhead. A common way for comparing the profitability of PoW compatible blockchains is via the Difficulty Adjusted Reward Index (DARI) [for], which is computed as:

$$\frac{R_i}{D_i} \cdot E, \tag{5.1}$$

where R_i is the (expected) reward for block i, D_i the difficulty of block i and E the exchange rate for the coin in some base currency (e.g. USD, BTC). Recall that R_i consists of transaction fees and the block reward which is constant for long periods of time. Therefore, it can

[3]The network adjusted time is the median timestamp of all the current times received by a node from its peers.

be seen that profitability is mostly impacted by changes in difficulty or the exchange rate. An economically rational miner aiming for short term profits, can thus engage in a strategy called *coin-hopping* [MCJ17, KL18, KKSK19], whereby the miner continuously redirects his computational power towards the most profitable cryptocurrency. On the other hand, long term profit-seeking miners allocate most of their computational power to the blockchain which they believe will have the highest valuation in the long run.

Depending on the distribution of hash rate across PoW compatible blockchains, hash rate fluctuations induced by coin-hopping behaviour impact the blockchains to different extents. For instance, on average 97% of the total `SHA-256` hash rate is concentrated in BTC, while the remaining 3% is distributed between BCH, BSV, and others.[4] As a result, fluctuations in the distribution of the total hash rate impact BCH and BSV significantly more than BTC.

5.2 Empirical Analysis of BCH's DA

In this section, we provide an empirical analysis on issues stemming from the DA that was employed in BCH in 2020 (at the time of this research). Note that this research was conducted at a time when `cw-144` was still the DA of BCH.

5.2.1 BCH's Difficulty Algorithms

Until November 2020, BCH had used two different difficulty adjustment mechanisms – previously a modified version of BTC's DA and shortly after a more responsive one called `cw-144` until it was abandoned in November 2020. To examine to what extent `cw-144` has been an improvement, we will also outline the workings of the initial DA.

[4]Data collected from `https://www.fork.lol`. Accessed: 2020-04-10

Emergency Difficulty Algorithm

When the BTC–BCH fork occurred on 1st August 2017, BCH kept BTC's PoW puzzle, while slightly adapting its DA. In BTC, the new difficulty D' is updated every 2 016 blocks based on the previous difficulty D, using the following formula:

$$D' = D \cdot \max\left(\min\left(\frac{2\,016 \cdot T}{T_A}, 4\right), \frac{1}{4}\right) \tag{5.2}$$

where T is the ideal inter-block time and T_A is the time it actually took to mine the last 2 016 blocks.

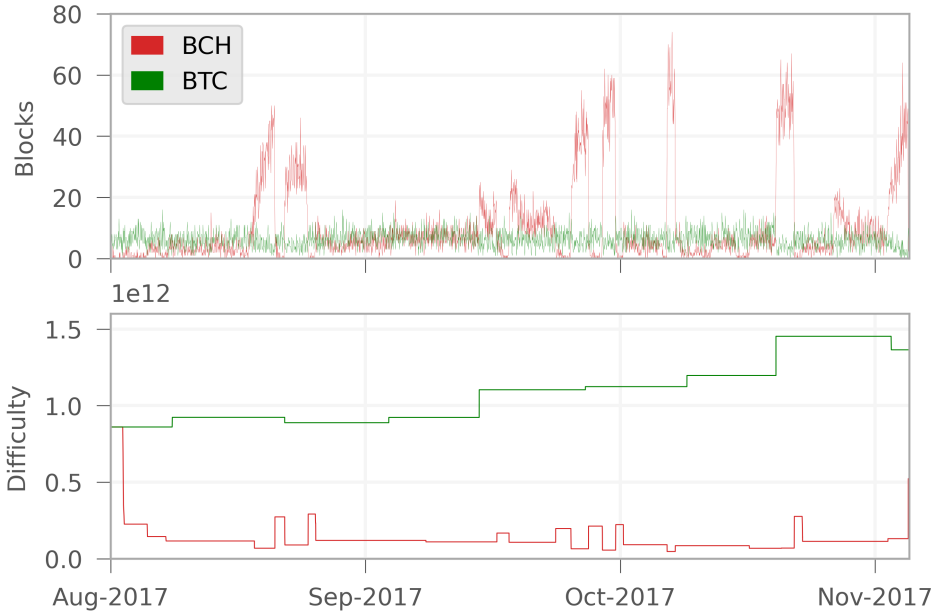

Figure 5.1: Number of blocks mined per hour (top) and block difficulties (bottom) for the period during which the Emergency Difficulty Algorithm was active.

As miners could engage in coin-hopping strategies and the loyal hash rate was expected to be much lower than BTC's, BCH developers foresaw that a scenario such as the one described in the last paragraph of Section 5.1.3 would arise. To ensure stable block throughput during large effluxes of hash rate, BCH resorted to the *Emergency Difficulty Algorithm* (EDA), whereby the difficulty would drop by 20% if the difference between 6 successive block timestamps exceeded 12 hours [AT19]. Therefore, BCH's first DA was a combination of BTC's DA and the EDA. However, it soon became apparent that this difficulty adjustment mechanism did not fulfill

its objective. Miners would stop mining BCH in order to cause consecutive 20% drops in the difficulty, which only adjusted back upwards every 2 016 blocks. Once the difficulty was sufficiently low, miners would switch back to mining BCH and produce many blocks at very low difficulty until the end of the 2 016 blocks window. As a result of this miner behaviour, from 1st August 2017 to 13th November 2017 a total of 9 947 more blocks were mined in BCH than in BTC, as shown in Figure 5.1.

BCH's Difficulty Algorithm `cw-144`

The combination of the 2 016 blocks window and the EDA was replaced on 13th November 2017 with a new DA proposed by BCH developer Amaury Sechet. BCH's DA, referred to as `cw-144`, attempted to increase responsiveness to both effluxes and influxes of hash rate by performing difficulty adjustments on a per-block basis.

The difficulty D of a new block is derived from the estimated hash rate, $\widehat{H}$, and the ideal inter-block time, T (i.e. 10 minutes). To this end, $\widehat{H}$ is computed using a simple moving average with a sample size of approx. 144 blocks. To mitigate situations when the block timestamps are out-of-order, the bounds of the sliding window over which the average is computed are derived using the median timestamp of 3 blocks. Thus, the block at which the window starts, B_{start}, is the block with the median timestamp out of blocks $144, 145$, and 146 in the past. Similarly, the window ends at block, B_{end}, with the median timestamp of the 3 most recent blocks. From these two blocks the DA computes W, the amount of work that was performed between these two blocks, as the sum of difficulties of all blocks in the interval $[B_{start}, B_{end}]$. The estimated hash rate is: $\widehat{H} = W/T_A$, where T_A is the actual time elapsed between B_{start} and B_{end}, capped in the interval from half a day to 2 days to prevent difficulty changing too abruptly. For completeness we give the full equation for the new difficulty:

$$D = \widehat{H} \cdot T = \frac{\sum_{i=start}^{end} \text{diff}(B_i)}{T_A} \cdot T \tag{5.3}$$

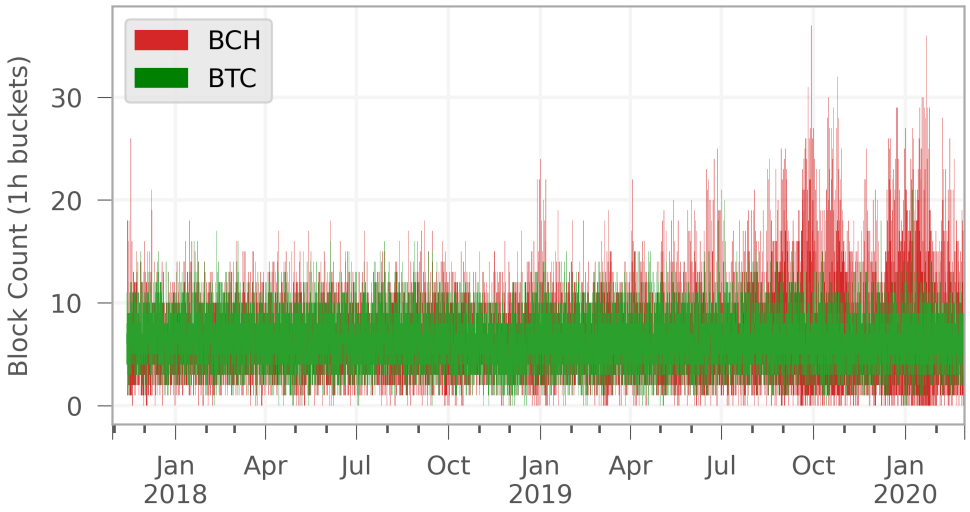

Figure 5.2: Number of blocks mined per hour in BTC and BCH since Amaury's DA deployment.

5.2.2 Oscillations in Number of Blocks Mined per Hour

As intended, `cw-144` achieves a daily average solve time of 10 minutes. This gives the superficial impression of performing well in terms of stable throughput, however certain patterns in the distribution of blocks within a day emerge.

From Figure 5.2 it can be seen that the oscillations in number of blocks mined per hour are notably more severe in BCH than in BTC. Especially during the later months, it is evident that BCH exhibited more 1 hour periods with either many blocks mined or none. As the number of blocks mined in an hour, K, should ideally follow a Poisson distribution with rate parameter $\lambda = 6$ blocks, we can compute the expected probability of mining exactly k blocks in one hour as:

$$P(K = k) = \frac{6^k}{e^6 k!} \tag{5.4}$$

We compare these ideal values with empirical results from BCH and BTC in Figure 5.3. For reference, it can be seen that in BTC the probabilities closely resemble those of a Poisson process. In contrast, BCH shows significant deviations from the ideal distribution during the period in which the EDA was active. After abandoning the EDA, BCH has indeed shifted towards the Poisson distribution, but a skew on the left and right tails remains. This is in line with the aforementioned observations of a more unstable transaction throughput in BCH, as

shown in Figure 5.2.

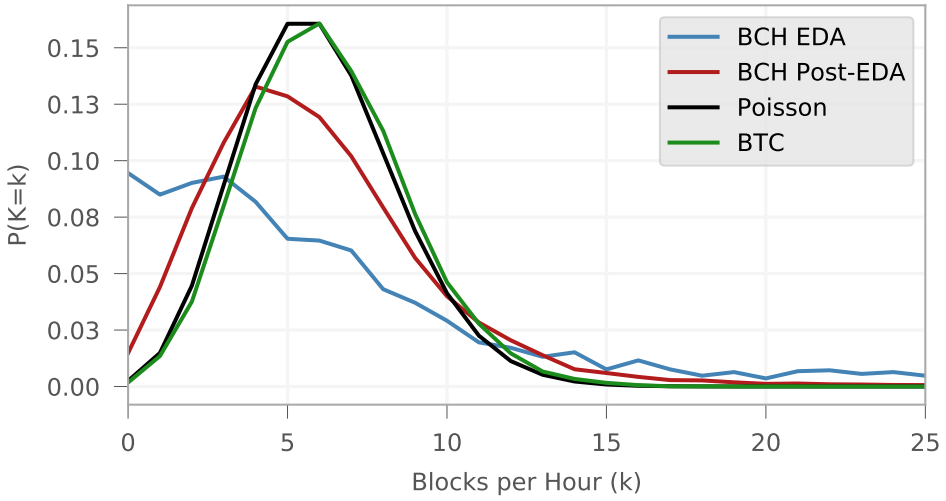

Figure 5.3: The probabilities of mining exactly k blocks in a one-hour period in BTC and BCH (pre and post EDA).

5.2.3 Positive Feedback Loop in Simple Moving Averages

The observed instability in transaction throughput can be explained by a positive feedback loop that stems from a combination of two factors: the use of a simple moving average and the miners' economically rational behaviour.

From formula (5.3) it is apparent that `cw-144` relies (in part) on the relationship of inverse proportionality between the time duration of the sliding window T_A and the estimated hash rate, $\widehat{H}$. The same relationship exists between the hash rate fluctuations and the solve times of newly mined blocks; i.e. solve times decrease when there is an increase of hash rate and they increase when there is a decrease. As new solve times are added to T_A, the result of these two relations is that $\widehat{H}$ is adjusted directly proportional to the actual hash rate change. However, the oversight of this DA is that using a simple moving average implies solve times falling off the window (subtracted from T_A) have an equal weight in the computation of $\widehat{H}$. Short solve times 144 blocks in the past cause a relative increase in T_A which yields a lower than expected $\widehat{H}$. Similarly, long solve times falling off the window imply a relative decrease in T_A and therefore produce a higher $\widehat{H}$. This influence constitutes positive feedback that results in correlation

between solve times 144 blocks apart. This is inherently problematic since mining is supposed to be a Poisson process where block arrivals are independent events.

The second factor that contributes towards the positive feedback loop is the miners' behaviour as they try to maximise profit by engaging in coin-hopping. For instance, assume BCH experiences an increase in profitability which incentivises a group of coin-hopping miners M_{CH} to switch their computational power towards BCH. This causes an increase in hash rate and consequently a series of blocks with short solve times. As the difficulty adjusts upwards, BCH's profitability drops causing M_{CH} to direct their hash rate towards more profitable chains. While the hash rate drops to its original value, the difficulty is now too high for the network, so a series of blocks with long solve times is produced.

This phenomenon of short solve times followed by long solve times is not intrinsically problematic, however, due to the positive feedback found in `cw-144` this pattern repeats continuously resulting in a positive feedback loop. This phenomenon has also been examined by [zaw19b, zaw19a, Too20].

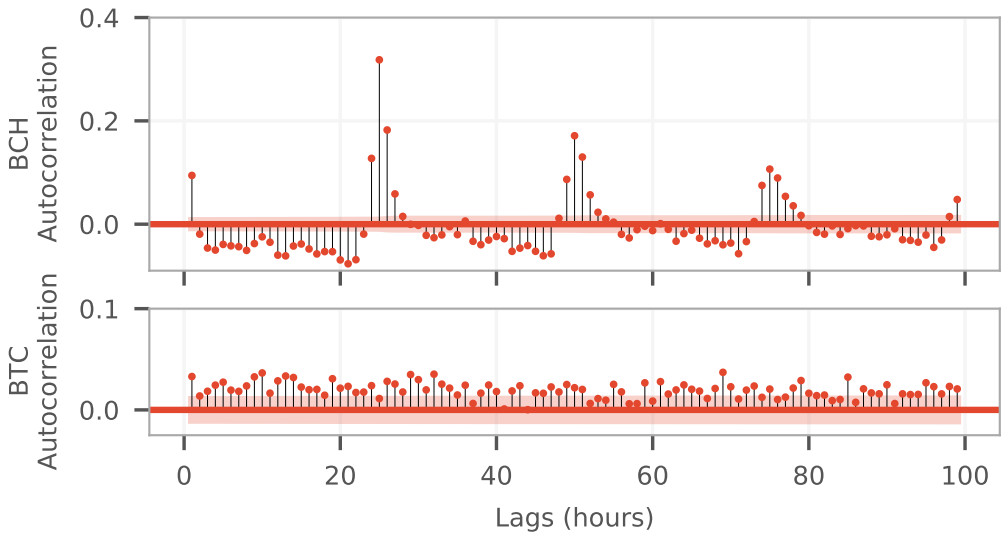

Figure 5.4: The autocorrelation in number of blocks mined per hour in BCH and BTC since Amaury's DA was deployed.

To investigate the extent of this cyclical phenomenon in BCH, Figure 5.4 compares the autocorrelation in the number of blocks mined per hour in BTC and BCH. As mining is supposed

to be a memoryless process there should not exist any significant autocorrelation which is what we see in BTC's graph. On the other hand, BCH has a significant positive autocorrelation immediately after 24 lags. Furthermore, in BCH weak positive correlation also persists after multiples of 24 lags. As a lag in this case represents a one hour interval, 24 lags represent the same expected duration as the sliding window of 144 blocks. Therefore, these empirical findings are in line with the positive feedback loop described.

5.2.4 Coin-hopping Incentives

Given the aforementioned issues related to difficulty adjustments, we attempt to assess the extent to which miners are incentivised to engage in coin-hopping and therefore contribute towards the formation of the positive feedback loop.

Deserts and Spikes

For the purposes of this analysis we define a *desert*, as a one hour interval during which at most 1 block is mined and a *spike* as a one hour interval during which 12 or more blocks are mined. Note, we have chosen these thresholds s.t. their probabilities are small and relatively comparable. Building on Equation (5.4), we compute the probabilities of mining at most k blocks per hour:

$$P(K \leq k) = \sum_{i=0}^{k} P(K = i) \tag{5.5}$$

Hence, we expect deserts and spikes to occur with a probability of $P(K \leq 1) = 1.74\%$ and $1 - P(K \leq 11) = 2.01\%$, respectively. We refer to a period which is neither a spike nor a desert as a *normal* period with $P(1 < K \leq 11) = 96.25\%$.

From Figure 5.5 it becomes apparent that not only are the expected likelihoods of deserts and spikes not achieved, but that the situation appears to be aggravating over time. For instance, over the last 6 months of the examined period, deserts and spikes occurred 13.5% and 12.7% of the time, respectively. By contrast, in BTC the respective percentages are 1.6% and 2.2%, which are significantly closer to the expected values.

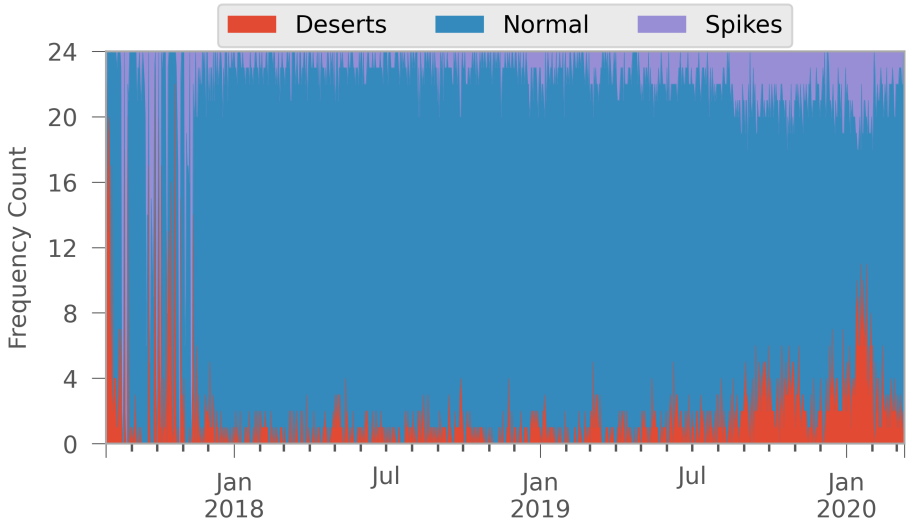

Figure 5.5: The number of one hour intervals classified as spikes, deserts, and normal periods in BCH.

Mining Profitability Comparison

We examine differences in the mining profitability of BCH and BTC by comparing the ratio of their DARIs (i.e. BCH DARI over BTC DARI) in Figure 5.6. In the long term, mining either coin is equally profitable as the average DARI ratio has a value of 1.0266. However, as the ratio frequently oscillates this incentivises miners to adopt a coin-hopping strategy. Notably, in the latter months, the oscillations become significantly more frequent and consistently reaching deviations of 10% and even 15% either in favor of BCH or BTC. These fluctuations are reflected in an increased number of spikes and deserts during the same period as can be seen in Figure 5.5.

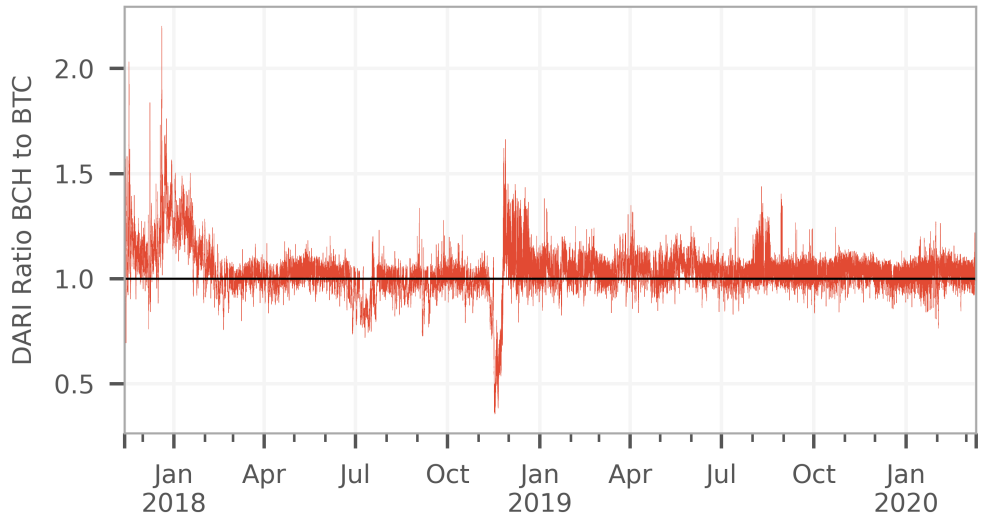

Figure 5.6: The minute average DARI ratio of BCH to BTC. Equal profitability is shown by the black line.

Miner Analysis

In order to examine the extent to which miners benefit from mining BCH, we analyse the block distribution for a set of high hash rate miners between 11$^{\text{th}}$ September 2019 and 11$^{\text{th}}$ March 2020. We deem this period relevant as the number of spikes and deserts is considerably higher than before.

In Table 5.1, we give data for the five largest BCH mining pools[5] (BTC.TOP, Antpool, BTC.com, ViaBTC, Huobi Pool) and three large miners without known identities, sorted by their respective shares of total blocks mined during the examined period.

[5]A mining pool allows multiple miners to combine their computational efforts and share the rewards. A pool can be seen as a single miner entity.

Miner	% of Total Blocks Mined in: Normal	Spikes	Deserts	Total
BTC.TOP	8.95 (51.59%)	8.28 (47.72%)	0.12 (0.69%)	17.35
Antpool	8.40 (79.78%)	1.84 (17.47%)	0.29 (2.75)%	10.53
`qp4ajq...`	2.36 (25.88%)	6.76 (74.12%)	0.00 (0.00%)	9.12
BTC.com	6.80 (82.03%)	1.18 (14.23%)	0.31 (3.74%)	8.29
`qqq9v3...`	2.21 (29.19%)	5.36 (70.81%)	0.00 (0.00%)	7.57
ViaBTC	5.91 (80.08%)	1.18 (15.99%)	0.29 (3.93%)	7.38
`qzkuv6...`	2.77 (53.68%)	2.39 (46.32%)	0.00 (0.00%)	5.16
Huobi Pool	2.47 (75.77%)	0.73 (22.39%)	0.06 (1.84%)	3.26

Table 5.1: Proportion of blocks mined by large miners during normal, spike and desert periods between block numbers 599 798 and 625 989.

Interestingly, out of the five largest mining pools, only BTC.TOP mined a similar amount of blocks during spikes and normal periods, while the remaining four pools mined on average 4.66% less blocks during spikes. This indicates that BTC.TOP is the only pool that successfully engages in coin-hopping by mining with higher hash rate during periods of lower difficulty. The other pools lose part of their block share to the coin-hopping miners. For instance, miners `qp4ajq` and `qqq9v3` obtained the third and fifth highest shares of blocks, while mining more than 70% of their blocks during spikes[6] and, perhaps rather impressively, none during deserts.

The extent of such coin-hopping behaviour can also be measured by analyzing the fluctuation in BCH's hash rate. The logarithmic scale chart from Figure 5.7 shows the hash rates of BTC and BCH over the last 6 months, estimated using a moving average of 6 blocks. While BTC's hash rate consistently oscillates from approx. 90 to 180 Exahashes per second, BCH's hash rate fluctuates from approx. 2 to 18 Exahashes per second. This means that BCH experiences periods when the hash rate increases even 9 times relatively to the baseline hash rate, which is inline with the results of Table 5.1.

[6]Full address of miners from Table 5.1 are given in Table A.1 from Appendix A.

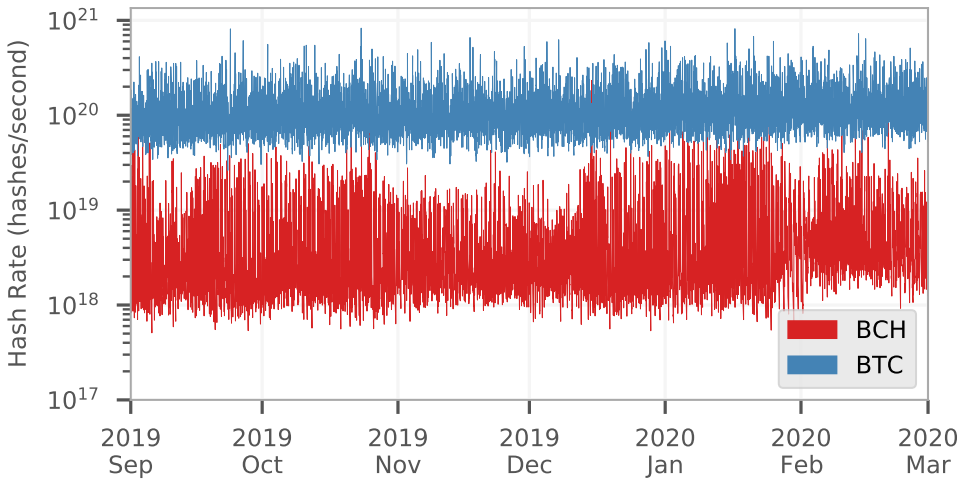

Figure 5.7: Estimated hash rates of BTC and BCH using a 6 block average.

5.3 Negative Exponential Filter Difficulty Algorithm

In this section, we mathematically derive a DA based on a negative exponential, low-pass filter. Although the initial formula may seem rather complicated, we can reduce it to a surprisingly elegant form which reveals several desirable properties, such as the lack of positive feedback.

To simplify mathematical computations and explanations we abstract the implementation details of the PoW target and mining difficulty. We model the target as a real number between 0 and 1 and ignore the maximum target requirement which is merely an implementation optimisation[7]. We refer to the inverse of the target as difficulty and note that it represents the expected number of hashes that need to be computed to obtain a valid PoW solution.

5.3.1 Difficulty Algorithm Requirements

DAs need to be reactive to hash rate fluctuations, especially in the case of coins where there might be regular influxes causing, e.g. a doubling or tripling of the overall hash rate. To ensure the difficulty adapts swiftly, the adjustment can be performed on a per-block basis, which is

[7]The maximum target was also the target of the genesis block and it was set to match the hash rate capabilities of the first miners.

indeed the design of `cw-144`. The DA we derive will maintain this property s.t. sudden hash rate fluctuations can be accounted for. Most DAs employ some kind of sliding window, considering only the most recent blocks. Intuitively, this approach is justified as the difficulty of older blocks is mostly a measure of the less evolved technology available at that time. However, for the reasons discussed in Section 5.2.3, we are interested in avoiding the use of a sliding window while still implementing its intention.

5.3.2 Mathematical Derivation

To satisfy the aforementioned requirements, we apply a negative exponential filter over all the block difficulties, weighing them based on time and a decay factor. For example, Figure 5.8 shows the effect of such a filter applied to 10 000 blocks from BCH. Notice how the more recent block difficulties are barely affected by the filter while the blocks far in the past will bring little contribution to the overall result. To obtain the estimated hash rate, we compute the weighted mean of difficulties over the full time span of the blockchain.

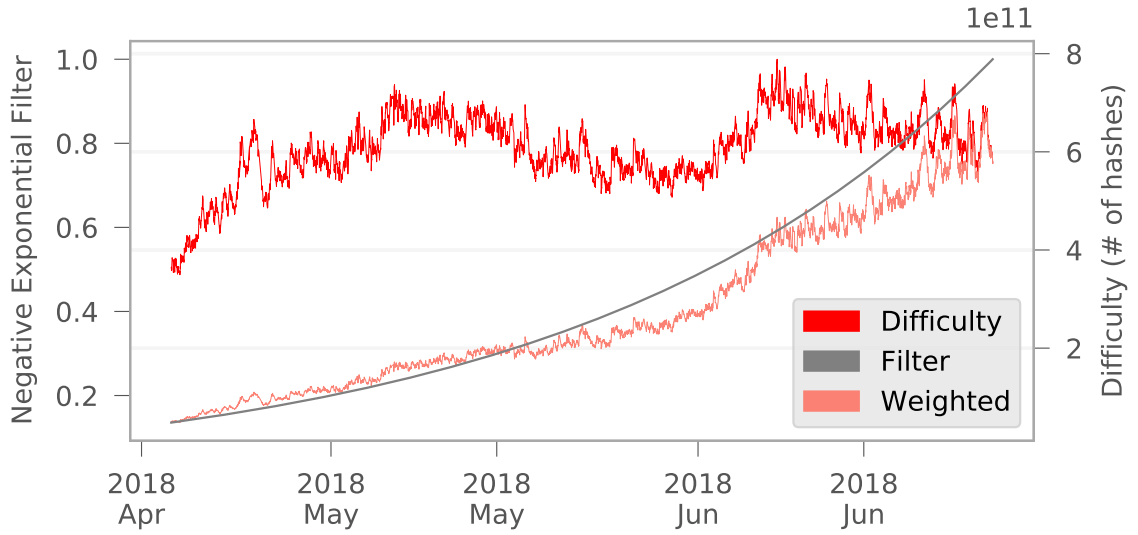

Figure 5.8: Difficulties of 10 000 blocks are filtered with a negative exponential to obtain the weighted difficulties.

The derived DA uses real time targeting, i.e. the difficulty of the block that is being mined dynamically adjusts based on the current time. Once a block is found, its timestamp allows

other miners to derive the block's difficulty and verify the PoW solution. To this end, we assume the algorithm operates in an idealised setting where miners never lie about block timestamps. We argue at length for the validity of this assumption in Section 5.3.5 and also give an alternative approach.

Throughout the remaining explanations, we make use of the following notation:

$$
\begin{aligned}
D_i &\leftarrow \text{difficulty of block } i \\
t_i &\leftarrow \text{time of block } i \\
st_i &\leftarrow \text{solve time of block } i : st_i = t_i - t_{i-1} \\
T &\leftarrow \text{ideal block solve time (e.g. 10 minutes)} \\
S &\leftarrow \text{decay/smoothing factor (see Section 5.3.4)} \\
\widehat{H}_i &\leftarrow \text{estimated hash rate at block } i
\end{aligned}
$$

For simplicity, index 0 refers to the block at which the new DA is deployed, while index n refers to the next block to be appended. Thus, t_n and $\widehat{H}_n$ represent the current time and network hash rate, respectively. NEFDA uses real time targeting (RTT), i.e. the difficulty D_n of the block that is being mined dynamically adjusts as time passes.

$$D_n = D_0 e^{\frac{t_0 + nT - t_n}{S}}$$

Considering RTT is not a popular technique, we argue at length for its safety in Section 5.3.5. In the remainder of this section, we show how this formula is derived from first principles.

Estimating Current Hash Rate

Difficulty algorithms are in the business of estimating the current network hash rate, $\widehat{H}_n$. As the actual function of hash rate cannot be known at any given time we rely on sampling when information is available, i.e. when blocks are mined. On average, the difficulty D_i represents the number of hashes computed throughout the interval $(t_{i-1}, t_i]$. Approximating that D_i hashes

are computed at time t_i we can estimate the current hash rate $\widehat{H}_n$ using exponential smoothing over the series of block difficulties, i.e. by taking their exponentially weighted average.

$$\widehat{H}_n = \frac{\sum_{i=0}^{n-1} D_i e^{\frac{t_i - t_n}{S}}}{\int_{-\infty}^{0} e^{\frac{x}{S}} dx} = \frac{1}{S} \sum_{i=0}^{n-1} D_i e^{\frac{t_i - t_n}{S}} \tag{5.6}$$

Difficulty Computation

Therefore, the difficulty D_n of the next block is:

$$D_n = T \cdot \widehat{H}_n = \frac{T}{S} \sum_{i=0}^{n-1} D_i e^{\frac{t_i - t_n}{S}} \tag{5.7}$$

$$= \frac{T}{S} \sum_{i=0}^{n-1} D_i e^{\frac{t_i - t_{n-1}}{S}} e^{\frac{t_{n-1} - t_n}{S}} \tag{5.8}$$

$$= e^{\frac{-st_n}{S}} \left(\frac{T}{S} \sum_{i=0}^{n-2} D_i e^{\frac{t_i - t_{n-1}}{S}} + \frac{T}{S} D_{n-1} \right) \tag{5.9}$$

$$= e^{\frac{-st_n}{S}} \left(D_{n-1} + \frac{T}{S} D_{n-1} \right) \tag{5.10}$$

$$= D_{n-1} \left(1 + \tfrac{T}{S}\right) e^{\frac{-st_n}{S}} \tag{5.11}$$

When unwinding the recurrence relation (5.11) all the way to D_0 we obtain:

$$D_n = D_0 \left(1 + \tfrac{T}{S}\right)^n \prod_{i=1}^{n} e^{\frac{-st_i}{S}} = D_0 \left(1 + \tfrac{T}{S}\right)^n e^{\frac{t_0 - t_n}{S}} \tag{5.12}$$

Correction

Notice that when $T \ll S$ we can approximate $1 + T/S \approx e^{T/S}$. In fact, this is actually a correction needed to mitigate the bias introduced when considering a discrete series of difficulties instead of the continuous function of hash rate. To prove this, we replace the constant term: $1+T/S$ with c and compute its value when the DA operates under a simple theoretical scenario. Specifically, we assume the hash rate remains constant for many blocks between m and n. Thus,

we expect the average rate of change in difficulty $\overline{R} = 1$, indicating that on average the difficulty does not change. We take the geometric mean of ratios of consecutive difficulties from block m to n and use Equation (5.12) with the c replacement:

$$\overline{R} = \sqrt[n-m]{\prod_{i=m+1}^{n} \frac{D_i}{D_{i-1}}} = \sqrt[n-m]{\frac{D_n}{D_m}} \tag{5.13}$$

$$= \sqrt[n-m]{\frac{D_0 c^n e^{(t_0-t_n)/S}}{D_0 c^m e^{(t_0-t_m)/S}}} = c \cdot e^{\frac{t_m - t_n}{(n-m)S}} \tag{5.14}$$

Assuming the DA is working correctly, the average solve time of blocks from m to n is $(t_n - t_m)/(n-m) = T$. Replacing in Equation (5.14) we obtain: $\overline{R} = 1 = c \cdot e^{\frac{-T}{S}} = 1$ which implies $c = e^{T/S}$.

Therefore, the correction is indeed justified and applying it in Equations (5.11) and (5.12), gives the following relative and absolute forms:

$$D_n = D_{n-1} e^{\frac{T - s t_n}{S}} \tag{5.15}$$

$$D_n = D_0 e^{\frac{t_0 + nT - t_n}{S}} \tag{5.16}$$

Interpretation: Clockwork Toy Time

Formula (5.16) can be interpreted by introducing the concept of "clockwork toy time" (`CTT`). We define the clockwork toy time as $\texttt{CTT} = nT + t_0$, where n is the height of the block currently being mined and t_0 is the timestamp of the block at which the DA was deployed. Therefore the proposed DA simply computes the difficulty of a new block by comparing the `CTT` with the current time. On average, if the `CTT` is ahead of the actual time it signifies that miners found more blocks than expected, so difficulty should be increased, which is what the formula accomplishes.

If we assume a scenario in which the hash rate increases and then remains constant it might not be immediately apparent why the difficulty stabilises. This is the case as the current time never catches up with the `CTT` because t_n is a summation of solve times tending on average towards T, while the `CTT` is a summation of T, so there will always be a lag between them. The

only reason the difficulty would drop is a decrease in hash rate, which on average would lead to longer solve times, therefore allowing the blockchain time to catch up with (and possibly exceed) the `CTT`.

5.3.3 Properties

History Agnosticism

The distribution of blocks in a given time period does not influence the difficulty of the block currently being mined. This property is desirable as block arrivals should be independent of each other so the difficulty of a block should not depend on the history of the chain. Equation (5.16) shows how the difficulty at time t_α depends only on the blockchain height, regardless of whether blocks were mined a long time in the past, in the last hour, or equally distributed in time.

Lack of Autocorrelation

Not only does this algorithm avoid the use of a sliding window, but the lack of autocorrelation is an emergent property entailed by history agnosticism. Sudden influxes or effluxes of hash rate may still produce temporary spikes or deserts, yet their duration will be much shorter. However, these will not create a positive feedback loop as the distribution of blocks in time has no influence on the future. Therefore, the inherent negative feedback present in NEFDA is the only force acting on solve times.

5.3.4 Smoothing Factor Considerations

The smoothing factor S has the function of configuring the reactiveness of NEFDA by setting the maximum rate of upward adjustments for the difficulty. More specifically, the difficulty can increase by at most a factor of e in S/T blocks. Depending on the requirements of the application, S should be chosen carefully: blockchains that are expected to experience large hash rate fluctuations on a regular basis (e.g. BCH), should aim for smaller values of S to

obtain a more reactive DA, while blockchains with a relatively stable hash rate (e.g. BTC) can choose larger values for S to reduce the difficulty's volatility. There is no direct relationship between the smoothing factor of an exponential moving average and the sample size of the simple moving average used in `cw-144`, as their operation is considerably different, but our simulations as well as other empirical studies [zaw19c] suggest that in order to obtain similarly stable difficulties the smoothing factor should be chosen to represent $(N + 1)/2$ blocks where N is the length of the sliding window used in simple moving averages. Applying this heuristic to BCH which has a sliding window of 144 blocks, suggests S should be set at approx. 12 hours. Note that selecting an appropriate value for S depends on the blockchain that utilises the underlying DA and should therefore be chosen with caution. Optimal selection of S may therefore be an avenue of future research.

5.3.5 Real Time Targeting Considerations

Real time targeting DAs assume miners have no incentive to report incorrect timestamps. To prove this assumption we compare NEFDA's RTT formulation with BCH's `cw-144` and argue that NEFDA reduces the incentives for timestamp manipulation. In `cw-144` reporting a dishonest timestamp, with a value in the future, would lower the difficulty for the next 144 blocks. This creates short term incentives for other miners to accept the dishonest block as they also benefit from the reduced difficulty even if they are not planning to be dishonest themselves. In contrast, NEFDA's history agnosticism implies that only the difficulty of the block with dishonest timestamp is affected, so there are no incentives for other miners to accept it. In fact, building on a dishonest block (B_i) implies mining towards a difficulty that is $e^{T/S}$ times higher than that of the previous block (B_{i-1}). Thus, a miner would only accept this block if it is willing to report an even higher timestamp to mitigate the increase in difficulty. This behaviour leads to an unstable chain as it could be replaced by a potentially shorter chain with more accumulated work (higher difficulty blocks), so honest miners would not risk accepting blocks with dishonest timestamps. Only an attack supported by a majority of the hash rate would be successful, which is no different than 51% attacks [Sza18, Bov11, zaw18] that are

currently possible in BCH or even BTC.

As an alternative solution, to completely remove possible issues arising from using RTT, we can replace any occurrence of a time t_i, with the median time of the 11 blocks preceding block i. As the MTP of 11 consecutive blocks is guaranteed to be ordered by consensus rules, this satisfies our requirement for monotonic time while not changing any of the assumptions existing DAs rely on. This does incur a delay of approx. 1 hour in estimating the current hash rate, but the proposed DA should still produce significantly better results than BCH's current DA. We note that a greater smoothing factor mitigates the drawbacks produced by the delay in hash rate estimation. Therefore, although we believe formulating the proposed DA using RTT is a valid alternative, the community can replace this using the MTP and configuring an appropriate smoothing factor.

5.4 Simulation

In this section, we empirically analyse the robustness of the Negative Exponential Filter DA by comparing it with BCH's current DA (described in Section 5.2.1). We perform this analysis by simulating the evolution of a blockchain under different scenarios of hash rate fluctuations. In particular, we focus on mimicking the behaviour of coin-hopping miners by adjusting the total hash rate in response to changes in profitability. Throughout this analysis we are mainly interested in the metrics we have already presented in the empirical analysis performed on BCH (see Section 5.2). For brevity, we abbreviate the Negative Exponential Filter DA derived in Section 5.3 with the acronym NEFDA and refer to BCH's current DA simply as BCH.

5.4.1 Setup

To simulate mining[8], we adjust the total hash rate to create various relevant scenarios that a DA could be exposed to.

[8]Note that these simulations do not use *PoolSim*, but rather a custom simulation setup focused on block generation times, difficulty and DAs.

For simulating `cw-144` and the MTP variation of NEFDA it suffices to simulate the blockchain evolution on a per-block basis. As we have knowledge of the actual hash rate we model the block solve times by using a random number generator that produces values distributed according to an exponential distribution with rate parameter $\lambda = H \cdot T/2^{256}$, where $T/2^{256}$ represents the success probability of one hash computation.

On the other hand, simulating NEFDA with RTT requires a more expensive computation as we have to update the target more frequently than at every block; we are satisfied with a per-second precision.

Through experimentation, we found that running the simulation for 100 000 blocks, corresponding to approx. two years of simulated time, is enough to clearly reveal any features of the metrics we are considering. We start simulations with a hash rate of 1 Exahashes per second, i.e. $H = 10^{18}$hashes/s, to maintain the scale of BCH's hash rate. Aiming for an ideal inter block time of 10 minutes, we initialise the blockchain with an appropriate target of $T = 2^{256}/(H \cdot 10 \text{ mins}) \approx 1.92^{56}$.

5.4.2 Modeling Miner behaviour

If all miners would focus on short term profit, then all the hash rate would be directed solely towards the chain with the highest profitability. However, in practice this scenario does not occur as miners have socio-political beliefs and might incur switching costs due to their mining configuration. As we are only interested in simulating miners' behaviour and not the relation between specific chains, we simplify profitability computations by only comparing the estimated DARI with its initial value and assuming a constant exchange rate as most `SHA-256` coins are highly correlated. Thus, we assume miners compute DARIs as the ratio between the average target of the last N_{DARI} blocks and the initial target of the blockchain. Although hash rate fluctuations appear in response to DARI oscillations, it remains unclear what exact switching logic miners employ. To this end, we believe the following 3 model of miners are general enough to capture the behaviour of any real miner.

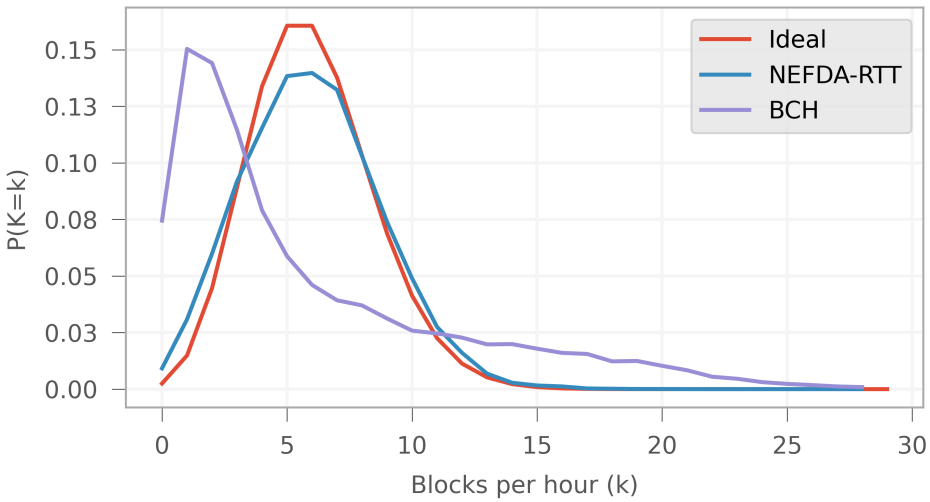

Figure 5.9: The probabilities of mining exactly k blocks in a one-hour period using NEFDA–RTT and `cw-144`.

Idealistic miners allocate all their hash rate to BCH regardless of how much profitability drops; they represent the baseline hash rate H_B.

Greedy coin-hopping miners allocate all their hash rate H_G towards BCH only when the DARI increases by at least 5%.

Variable coin-hopping miners allocate part of their total hash rate H_V in relation to the current profitability. Although this relation is not clear in reality, we consider a model based on the logistic curve. The intention is to emulate both the initial stage when the hash rate increases exponentially as miners realise the advantage in profitability, and the later stage when the hash rate influx gradually slows down. The model directs all the hash rate away from or towards BCH, if drops or increases in profitability larger than 15%[9] occur. Otherwise, a variation x between -15% and 15% leads to a contribution of $H = H_v/(1 + e^{-6/0.15 \cdot x})$ towards the total hash rate.

5.4.3 NEFDA–RTT vs. `cw-144`

We start by comparing RTT-based NEFDA with `cw-144` when $H_V = H_G = 4 \times H_B$ and $N_{\text{DARI}} = 6$ blocks. A very brief analysis of the average solve times: 599.97 s for NEFDA–

[9] We have chosen this value based on data presented in Section 5.2.4.

RTT and 604.34 s for `cw-144`, already reveals how NEFDA–RTT achieves a more appropriate value. Furthermore, the probability distribution of the number of blocks per hour produced using NEFDA–RTT and `cw-144` are compared to the (ideal) expected values in Figure 5.9. The distribution obtained with NEFDA–RTT shows very minor deviation from the ideal plot, probably as a result of the extreme scenario simulated. At the same time, `cw-144` shows significant skew on both tails of the distribution, suggesting that both deserts and spikes are produced. Analyzing the per-hour block count reveals a much larger standard deviation in `cw-144` than in NEFDA–RTT (see Figure 5.13). Indeed, deserts and spikes occurr 22.4% and 17.5% of the time when using `cw-144` while only 3.6% and 2.7% when using NEFDA–RTT.

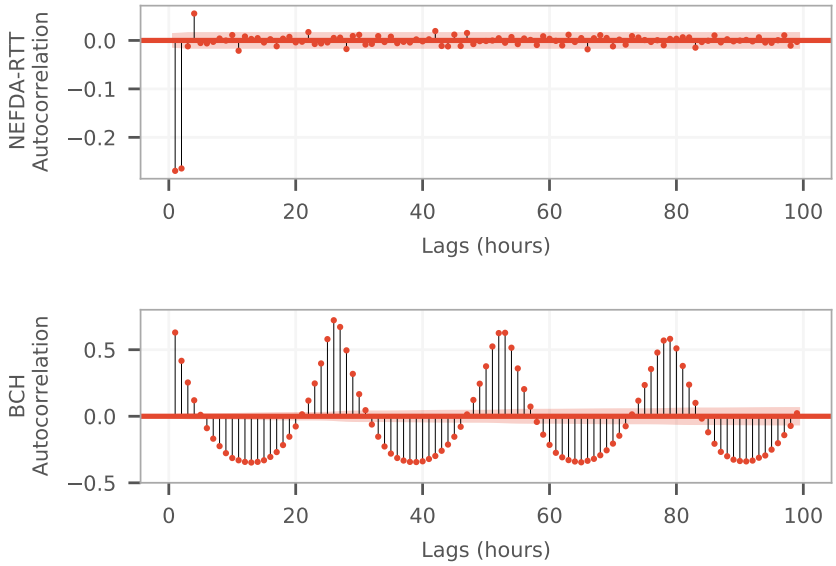

Figure 5.10: The autocorrelation in number of blocks mined per hour in NEFDA–RTT (top) and BCH/`cw-144` (bottom).

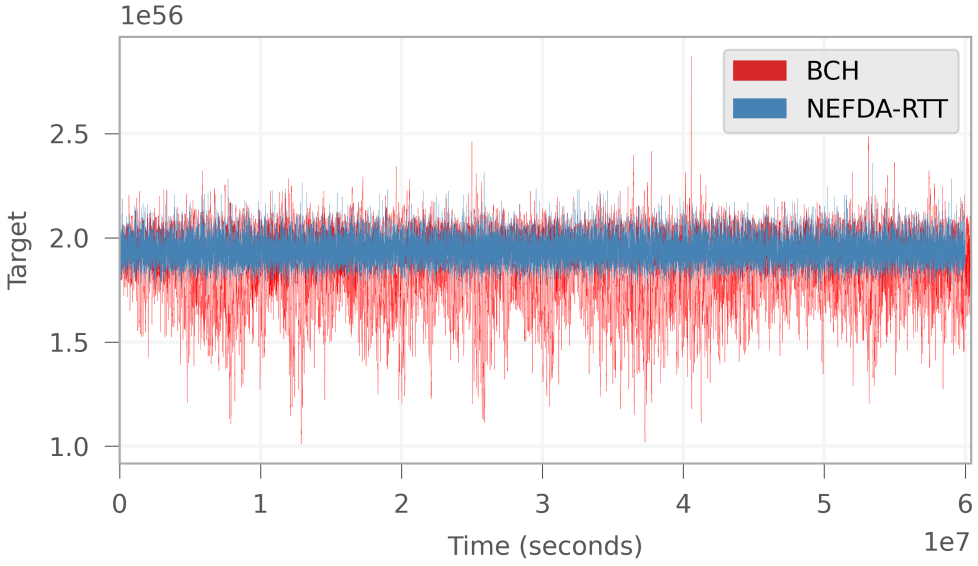

Figure 5.11: The targets per hour for NEFDA–RTT and BCH/`cw-144` in a coin-hopping simulation of 100 000 blocks.

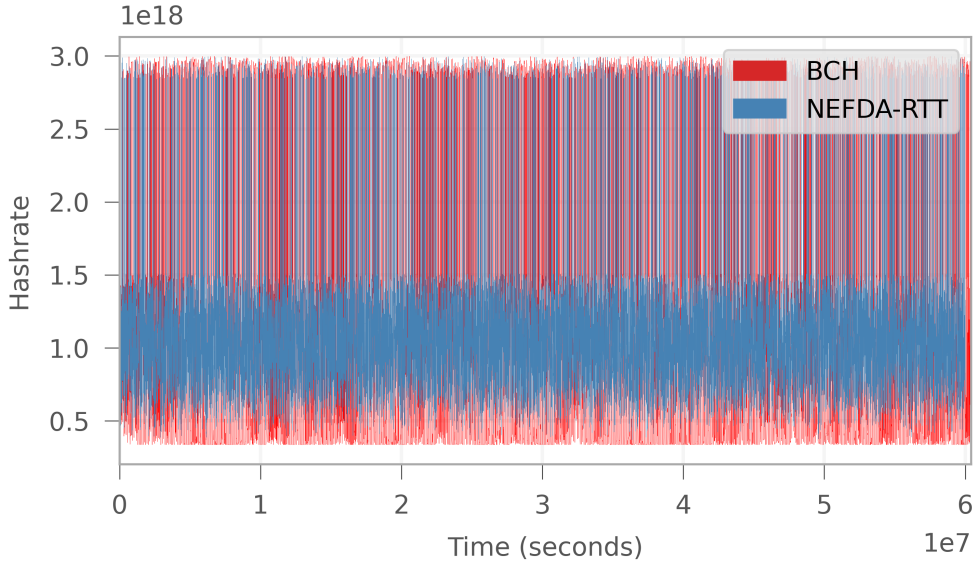

Figure 5.12: The hash rates per hour for NEFDA–RTT and BCH/`cw-144` in a coin-hopping simulation of 100 000 blocks.

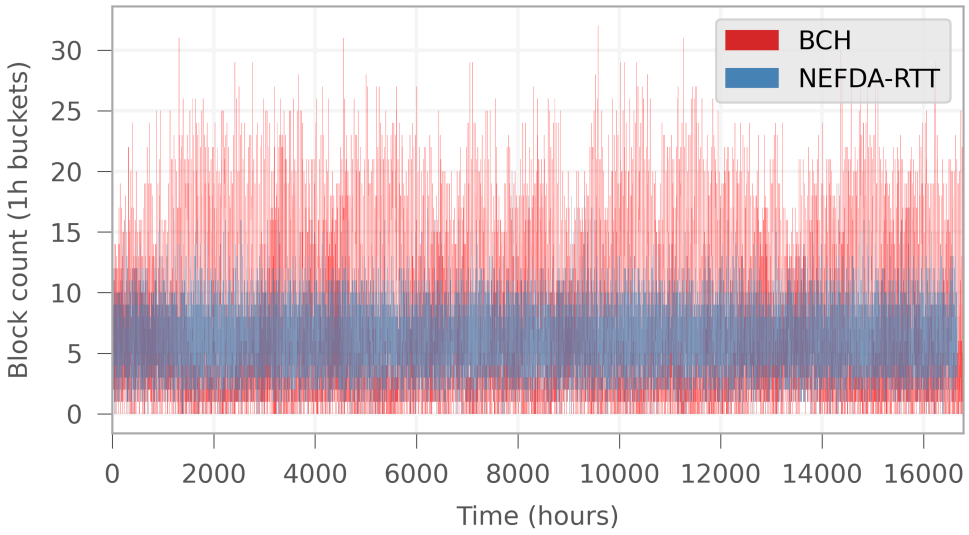

Figure 5.13: The blocks mined per hour for NEFDA–RTT and BCH/`cw-144` in a coin-hopping simulation of 100 000 blocks.

As expected, the reason for this massive discrepancy is the positive feedback loop present in `cw-144`. Figure 5.10 shows the significant amount of positive correlation that appears at multiples of W (the size of BCH's sliding window). Interestingly, `cw-144` also shows negative correlation between blocks that are $W/2$ apart, indicating that there is a delay of 12 hours in hash rate estimation. On the other hand, NEFDA–RTT shows negative correlation between neighboring hour-buckets indicating that the DA rapidly responds to sudden hash rate fluctuations. By adjusting the target more quickly the DARIs oscillations are limited and miners are less incentivised to abandon mining (see Figures 5.11, 5.12, 5.13 for target, hash rate and DARI results, respectively).

5.4.4 Smoothing Factor Trade-offs

To examine the influence of the smoothing factor over the performance of NEFDA, we model a slightly more extreme environment by increasing the total hash rate of variable miners: $H_V = 6 \times H_B$. Simulations are run using NEFDA–RTT with 3 smoothing factors that are multiples of the ideal inter block time, in order to represent 36, 72 and 144 blocks. An initial analysis of the average solve times: 600.0015 s for NEFDA–36, 599.9835 s for NEFDA–72 and

599.9895 s for NEFDA–144 shows how NEFDA in general copes much better than `cw-144` even in more extreme environments. However, we do observe slight improvements in the targeting of the ideal average of 600s as the smoothing factor is increased.

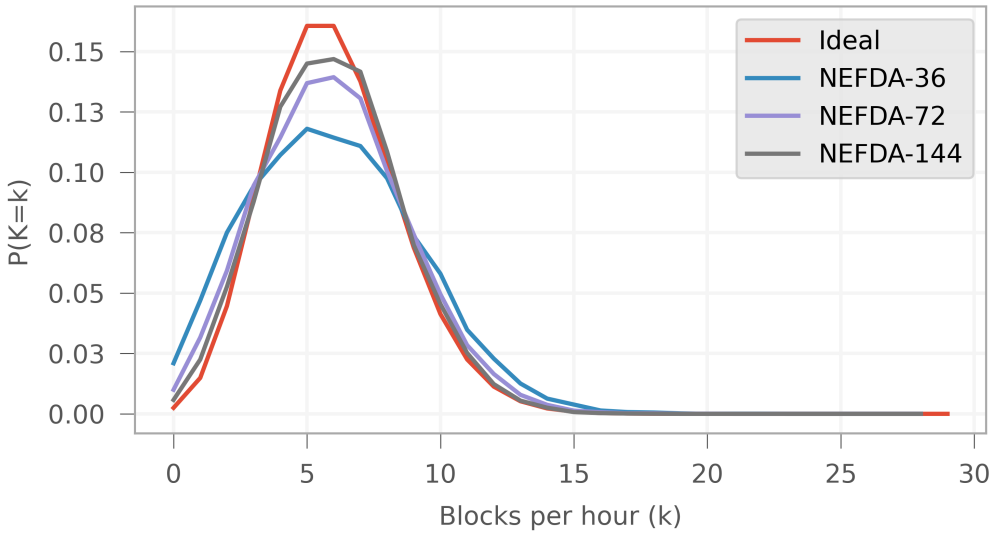

Figure 5.14: The probabilities of mining exactly k blocks in a one-hour period using various smoothing factors for NEFDA–RTT.

Figure 5.14 reveals that all distributions have an appropriate center of mass, but lower smoothing factors slightly flatten the curve skewing the distribution from the ideal values. These small discrepancies affect the proportion of deserts: 6.77%, 4.14%, 2.81% and spikes: 4.84%, 3.06%, 2.14% (values are given in order for each of the smoothing factors considered: 36, 72 and 144 block). As shorter smoothing factors attribute greater relevance to recent blocks the algorithm risks being too reactive, and therefore over or underestimating the current network hash rate. As shown in Figure 5.15, this effect translates in more volatile targets, especially when underestimating the hash rate due to long block solve times. In turn, this leads to more volatile profitability which encourages coin-hopping behaviour.

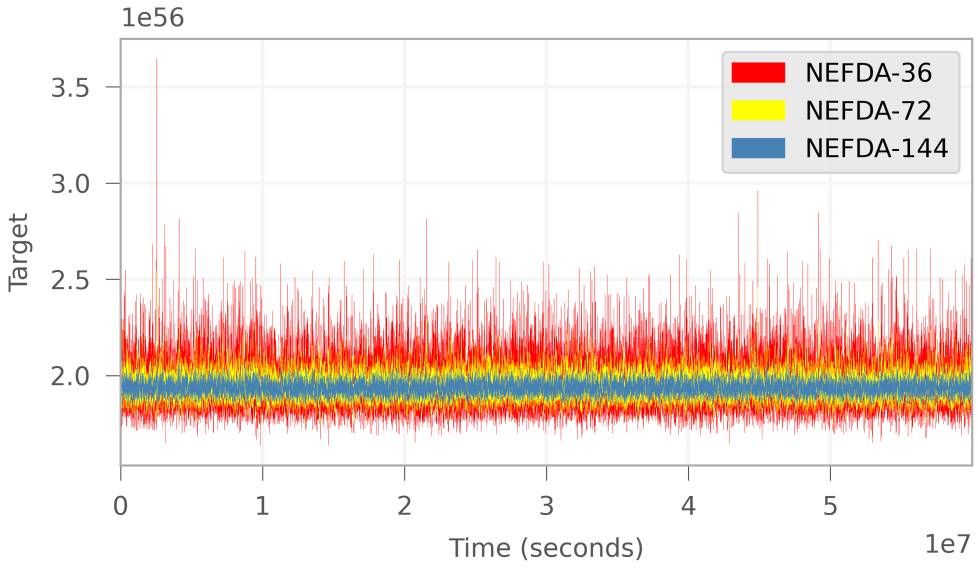

Figure 5.15: Comparison of the targets produced using different smoothing factors in NEFDA.

On the other hand, greater smoothing factors are not always desirable. Figure 5.16 shows the evolution of targets for each of the three variants of NEFDA when simulating a scenario in which the hash rate increases exponentially without any coin-hopping behaviour. Although the lower smoothing factors still imply a more volatile target, the average solve times are now: 599.5518s for NEFDA–36, 599.1267s for NEFDA–72 and 598.2724s for NEFDA–144, which imply that the more reactive nature of NEFDA–36 and NEFDA–72 is desirable under these conditions.

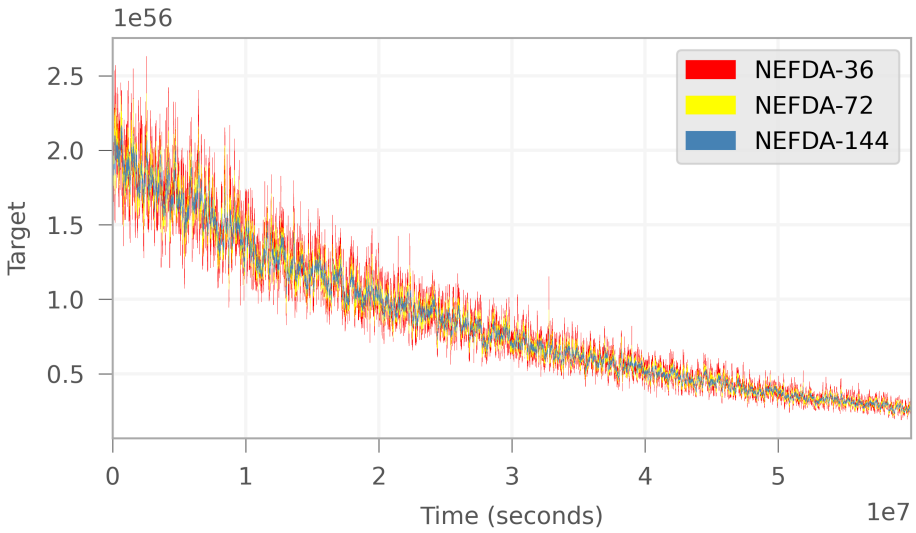

Figure 5.16: Comparison of the targets produced using different smoothing factors in NEFDA under exponentially increasing hash rate.

5.4.5 Median Time Past Considerations

Lastly, we analyse the effects of using the MTP variant of NEFDA, by simulating it under the first scenario ($H_V = H_G = 4 \times H_B$ and $N_{\text{DARI}} = 6$ blocks.) As mentioned in Section 5.3.5, the MTP of the last 11 blocks introduces a lag of approx. 1 hour in hash rate estimations. A small smoothing factor implies NEFDA's current hash rate estimate is mainly based on the most recent blocks. Due to the lag introduced by the MTP these recent blocks may not have their difficulties updated accordingly and a positive feedback might be introduced in the computation. However, unlike with `cw-144`, where the positive feedback leads to a perpetuating loop due to the use of equal weights and a sliding window, NEFDA's positive feedback quickly diminishes as the weights of the problematic blocks fade in time (see Figure 5.21).

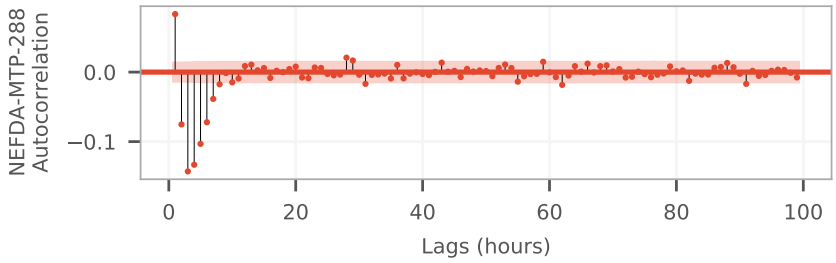

Figure 5.17: The autocorrelation in number of blocks mined per hour for NEFDA–MTP with smoothing factors 288 in a coin hopping simulation of 100 000 blocks.

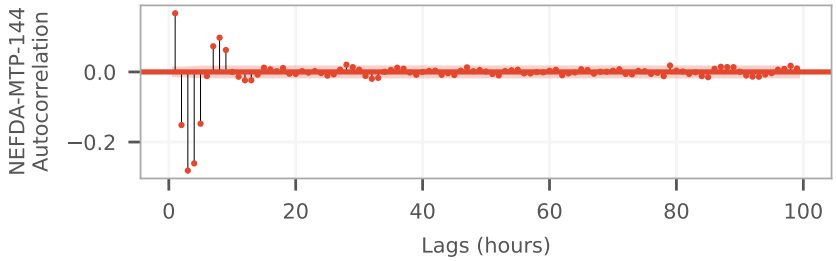

Figure 5.18: The autocorrelation in number of blocks mined per hour for NEFDA–MTP with smoothing factors 144 in a coin hopping simulation of 100 000 blocks.

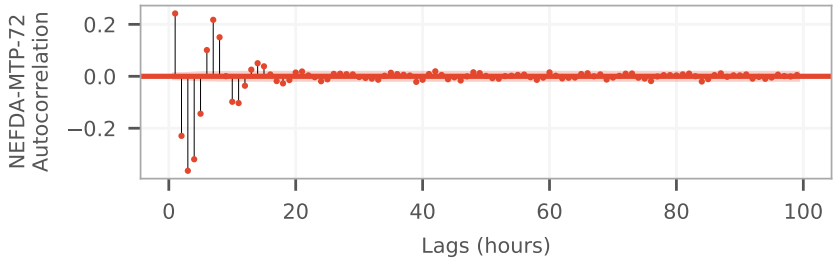

Figure 5.19: The autocorrelation in number of blocks mined per hour for NEFDA–MTP with smoothing factors 72 in a coin hopping simulation of 100 000 blocks.

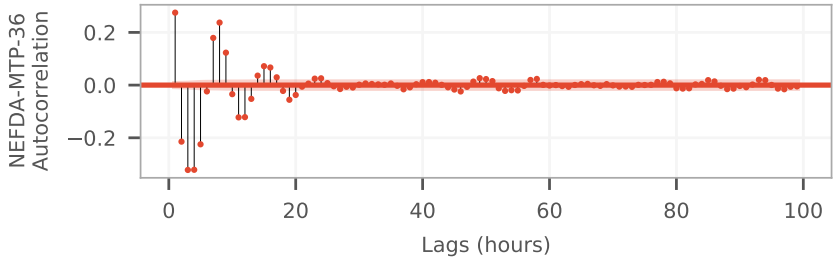

Figure 5.20: The autocorrelation in number of blocks mined per hour for NEFDA–MTP with smoothing factors 36 in a coin hopping simulation of 100 000 blocks.

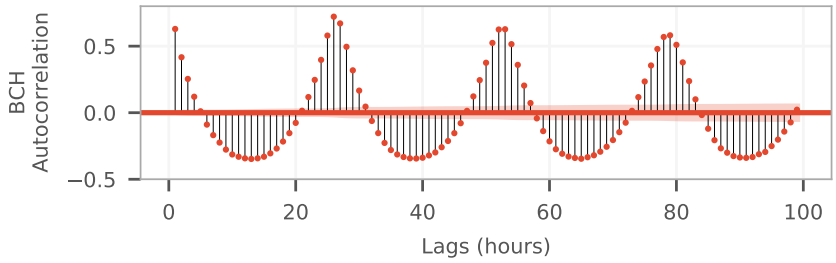

Figure 5.21: The autocorrelation in number of blocks mined per hour for `cw-144` in a coin hopping simulation of 100 000 blocks.

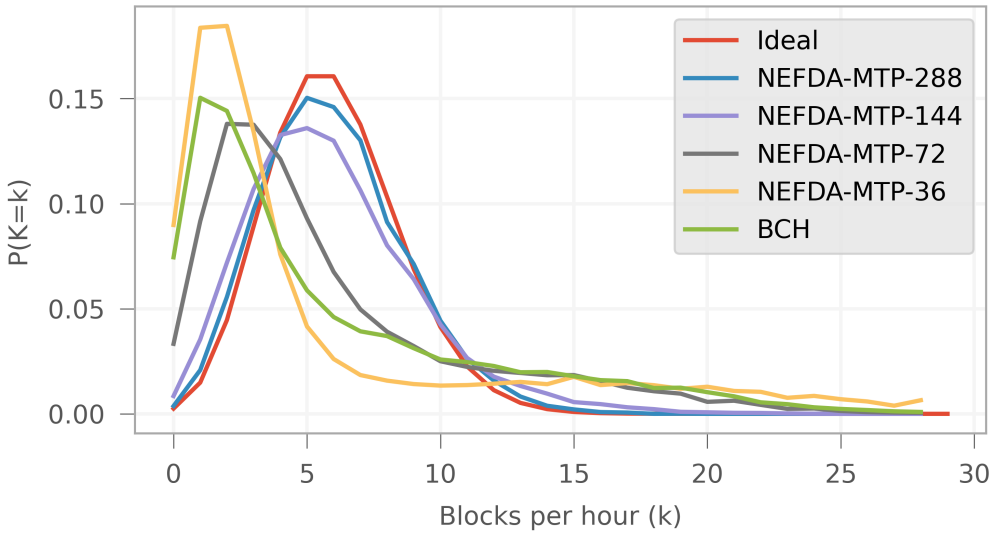

Figure 5.22: The probability distribution of exactly k blocks being mined during a one hour period using various smoothing factors for NEFDA-MTP, compared to the ideal values.

This effect is reflected in skewed block distributions (see Figure 5.22). Notice how higher smoothing factors mitigate the use of MTP. Therefore, if a community decides in favor of MTP, we suggest assuming the costs of a less reactive DA and selecting a smoothing factor much larger (e.g. 20 times) than the MTP's window size.

5.5 Related and Follow-up Work

In this section, we will discuss related work at the time of this research, as well as discuss follow-up work that is relevant to this chapter's contributions, yet was released after this chapter's contributions had been published.

5.5.1 Related Work

The most extensive body of difficulty algorithm research has been done by the pseudonym zawy12, who provides a comprehensive overview of various difficulty algorithms in [zaw19c]. He also examines the difficulty instabilities in BCH in [zaw19a]. Regarding our proposed DA, zawy12 simulates the performance of two algorithms also based on exponential filters: ASERT [Lun19] and EMA [zaw19d], which is an approximation of ASERT that avoids the computation of exponentials. We have become aware of ASERT which is essentially equivalent to NEFDA, only after receiving the unpublished work of Mark B. Lundeberg from zawy12[10]. Our additional contribution consists of the mathematical derivation of this algorithm, an outline of desirable properties, and motivation for the correction. In parallel work, an explanation of how a DA based on an exponentially weighted moving average could mitigate the difficulty instabilities in BCH was provided by [Too20].

The following are related pieces of work in the broader contexts of difficulty adjustment algorithms as well as coin hopping. In [Kra16] the author shows how Bitcoin's difficulty algorithm causes shorter block solve-times in the case of an exponentially hash rate growth and defines an alternative model which achieves desired average block times in the long-run, yet is subject to increased solve time fluctuations. A stochastic model is presented in [FM18], where the difficulty target is modeled as a random variable that is a function of previous block times. The difficulty adjustment problem is addressed from a feedback control engineering perspective in [HK17], where the difficulty is adjusted on a per-block basis using a non-linear feedback controller based on a moving average filter of recent block timestamps for ensuring stable block solve times. Bobtail is an alternative difficulty algorithm presented in [BL17], which reduces block solve-time variance, yet comes at the cost of requiring significantly larger block headers. In [BTL19] a new difficulty algorithm is presented based on "bonded mining", whereby a miner has to put up collateral and commit to mine at an offered hash rate over a period of blocks.

An investigation into the decision-making process driving miner behaviour during times of BCH's EDA is presented in [AT19]. A game theoretic framework of miners switching between

[10] At the time of this research, an unpublished version of what later became [Lun20] was sent to us.

PoW-compatible blockchains based on difficulty is proposed in [KKSK19]. The authors further show that BCH experienced a lack of loyal miners prior to the introduction of `cw-144`, which temporarily undermined the security of the system. The consequences of miners switching under `cw-144` remained unexamined.

5.5.2 Follow-up Work

This subsection discusses some work that was published after our contributions were published. We deem this relevant in the context of the contributions presented in this chapter.

Two proposals [fTCL20, Sec20] were put forward to replace `cw-144`. The proposal that essentially was the one that replaced `cw-144` was ASERT[fTCL20], which was implemented on November 15, 2020, and cited NEFDA (the citation was made to an earlier pre-printed version of the published paper).

While most existing DAs primarily focus on ensuring stability in block creation within specific timeframes, the authors of [DYQ+22] introduce a new DA with the objective of stabilising orphan rates among recently generated blocks. They argue that the orphan rate serves as a more informative metric, as a high orphan rate indicates weak blockchain security and wasted honest hash power, while a low orphan rate signifies inadequate blockchain efficiency and unnecessary delays in waiting for other blocks. The authors identify a security flaw in an orphan-rate DA presented by [ZZW+20], whereby attackers can continuously generate orphan blocks, misleading miners into incorrectly increasing the difficulty level. To rectify this vulnerability, the authors propose an enhanced DA that considers the variation in orphan rates during historical slots. They incorporate a smooth exponential decay effect into the current difficulty adjustment mechanism.

5.6 Conclusion

In this chapter, we first provide a case study on BCH's DA and show how the behaviour of economically rational miners can lead to severe instabilities in throughput as a consequence of a positive feedback loop stemming from `cw-144`. The cyclical pattern in block solve times skews the distribution of the number of blocks per one-hour intervals therefore having a negative impact over the transaction throughput. In order to mitigate periods of undesired (either too low or too high) throughput, we model NEFDA, a DA which does not lead to the formation of a positive feedback loop and can cope effectively with sudden hash rate fluctuations. We explain how NEFDA exhibits desirable properties in the form of history agnosticism and lack of any significant autocorrelation. Additionally, we show how NEFDA is configurable in the level of responsiveness. Through simulations, we demonstrate how NEFDA outperforms `cw-144` in terms of reducing target volatility and in turn high variations in block solve times. Furthermore, we show how to mitigate drawbacks introduced by the MTP variation of the NEFDA, by configuring the smoothing factor. Therefore, NEFDA constitutes a viable alternative for both large and small blockchains (in terms of baseline hash rate) when configured appropriately.

In the broader context of this thesis, this chapter has successfully shown how implementing some arbitrary mechanisms to control critical system features, such as the difficulty levels in the Proof-of-Work, can give rise to undesirable incentives and thereby undermine the desired workings of the system. The long and rather troublesome journey of developing and selecting an appropriate difficulty algorithm in Bitcoin Cash has shown that designing, maintaining and transitioning blockchain systems requires a sophisticated set of tools that allow researchers and engineers to formally study and simulate novel mechanisms in different environments (e.g., with different network hash rates, coin-hopping, etc.) to avoid any potential negative effects to the stability of the network. We have demonstrated in this chapter how such tools may be used by identifying and carefully constructing the desired properties of a favorable difficulty algorithm, which we then mathematically derived and subsequently simulated under different network assumptions.

Chapter 6

(Un)Stable Transaction Throughput: Leveraging the Cyclicality of Transaction Fees

Since the introduction of Ethereum and its virtual machine, participants have been able to create so-called *smart contracts*, i.e. programs that encapsulate the logic for governing funds. As these contracts have to be executed by all participating nodes in the Ethereum network, the sender of a transaction has to pay for the computational cost of execution in units of *gas*. The amount of gas to be paid by the sender of a transaction depends on the complexity of executing a smart contract's logic. Additionally, the sender is required to specify the *gas price*, which he will have to pay per unit of consumed gas. The product of the gas cost and price determines the transaction fee, which is received by the miner who includes the transaction in a block. Hence, setting an appropriate gas price is critical for having a transaction included in a timely manner. While Ethereum employs a hard-coded and transparent gas cost model, there does not exist any embedded mechanism for computing how much a sender of a transaction should pay per unit of gas. The gas price is instead determined by the supply and demand for computational resources. Therefore, choosing an optimal gas price can be challenging, as underpaying likely results in a transaction not being included by miners, whereas overpaying

leads to avoidable costs.

The most widely used gas price prediction mechanism is implemented by the popular Ethereum client Geth [Git20b]. This and comparable mechanisms only use recent gas prices and merely aggregate past data to heuristically recommend the gas price for a transaction.

In this chapter, we present a novel approach for gas price prediction, motivated by the empirical analysis of a period of $522,213$ blocks. We find significant seasonality in the gas price data, suggesting that this can be predicted using a machine learning model. We propose the use of Gated Recurrent Units [CvMG+14] as these have been shown to be suitable for capturing such patterns. Consequently, we design an algorithm for choosing the gas price for a transaction, which leverages the predictions of our model while allowing to specify the transaction's urgency. Our evaluation on real-world data shows that the proposed approach significantly outperforms the most widely-used Ethereum client Geth [eth20c].

Note that this research was done when Ethereum still used Proof-of-Work as its consensus mechanism and prior to the EIP-1559 update.

Contributions

- We present a comprehensive empirical analysis of the Ethereum gas price over a period of three months and identify seasonal patterns in the data,

- We propose a deep-learning based model to predict the gas price and combine this with a novel algorithm for recommending the gas price for a transaction,

- We evaluate our model on real-world data and show that it outperforms the most widely used gas price recommendation approach, resulting on average in costs savings of more than 50% while only incurring an inclusion delay of 1.3 blocks compared to Geth.

6.1 Empirical Analysis

In this section, we empirically analyse Ethereum block data to develop a better understanding of the gas price behaviour. We use data from the period of 1 October, 2019 to 31 December, 2019, which amounts to a total of 522, 213 blocks. When comparing mean, minimum and maximum gas prices averaged over 3 hour intervals during this period, we can see in Figure 6.1 that substantial spreads exists in the gas price. More specifically, the maximum gas price exceeds the minimum gas price by an order of magnitude for the entire period. The gas

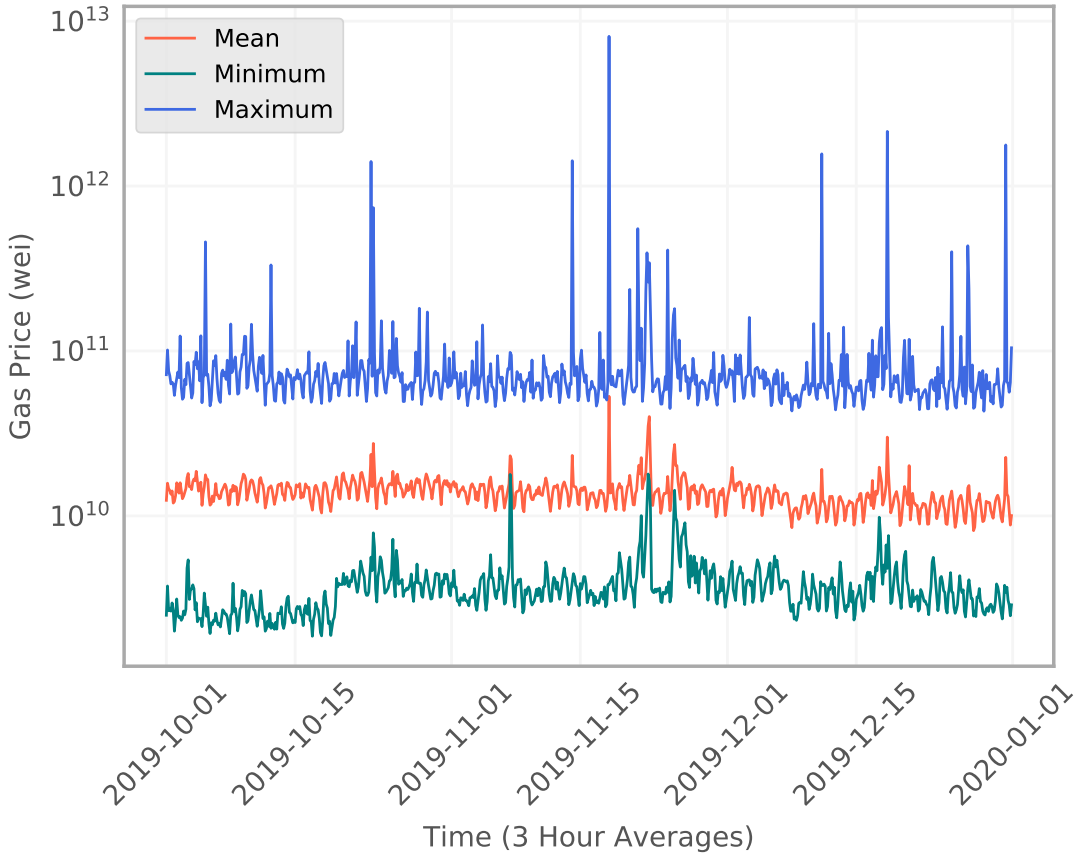

Figure 6.1: The mean, maximum and minimum gas price averaged over 3 hour intervals from block 8, 653, 173 (1 October, 2019) to 9, 193, 265 (31 December, 2019).

price volatility throughout the examined 522, 213 blocks is further indicated by the standard deviation of the average gas price, which is 46.4645 Gwei at an average gas price of 13.9598 Gwei, as shown in Table 6.1. The same can be said about the average gas utilisation per block. Figure 6.2 shows the cross-correlations between the average gas price, maximum gas price, minimum gas price, number of transactions and gas utilisation per block. Surprisingly,

Number of blocks:	522,213
Mean gas price:	13.9598 Gwei
Median of average gas price:	10.3260 Gwei
Standard deviation of average gas price:	46.4645 Gwei
Mean gas utilisation:	79.36%
Standard deviation gas utilisation:	32.00%

Table 6.1: Mean, median and standard deviation of average gas price per block, as well as mean and standard deviation of gas utilisation per block from block 8,653,173 (1 October, 2019) to 9,193,265 (31 December, 2019).

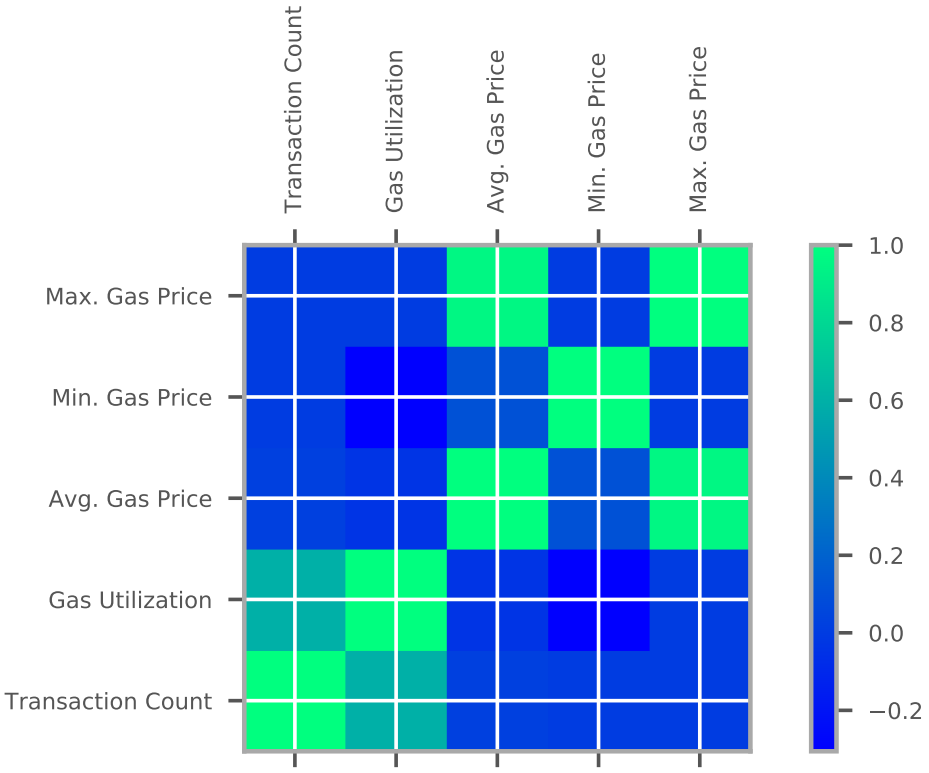

Figure 6.2: Correlation matrix for the average gas price, maximum gas price, minimum gas price, number of transactions and gas utilisation per block.

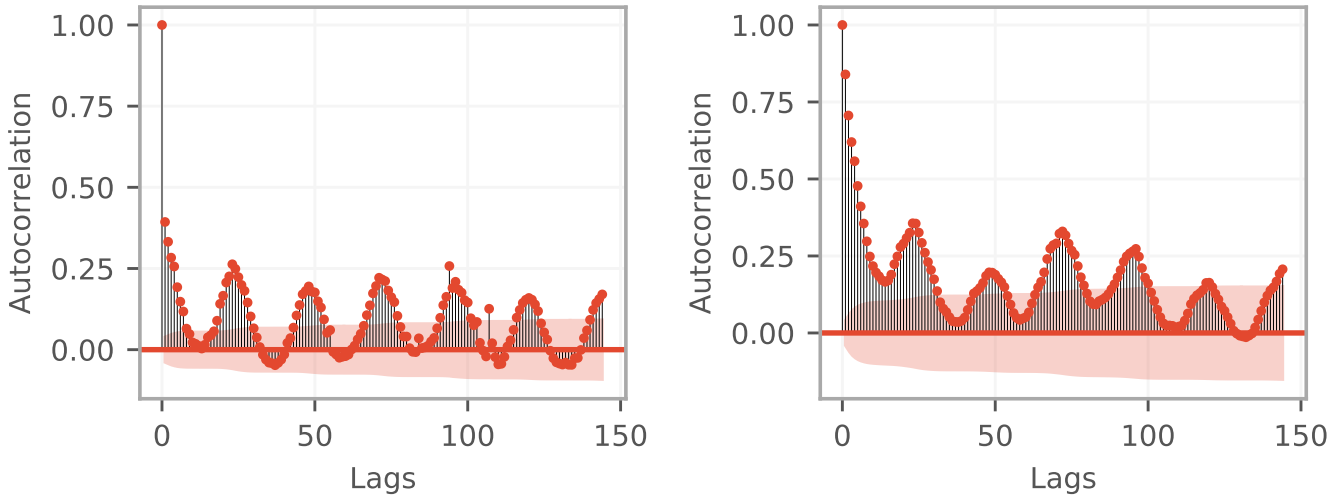

Figure 6.3: Autocorrelation function (ACF) plot of mean (left hand side) and minimum (right hand side) gas prices averaged over one hour periods for 144 lags (hours).

the average gas price and utilisation are not correlated. In fact, the average gas price is only significantly correlated with the maximum gas price. The gas utilisation is only correlated with the transaction count. However, apart from these two correlated pairs, the remainder of the variables are not significantly correlated.

To investigate the presence of seasonality in the data, we examine the autocorrelation of each variable on a per block and hourly basis. Most interestingly, we find that even though the gas price does not exhibit any significant seasonality on a per block basis, there does exist seasonality when looking at the gas price averaged over one hour intervals, as indicated by the autocorrelation in the left plot of Figure 6.3.

It can be seen that especially for a lag of 24 hours significant seasonality can be found in the data, which could be linked to different time zones of the countries where most transactions are conducted. This seasonality can be found to an even greater extent in the autocorrelation of the minimum gas price averaged over one hour intervals. The presence of seasonal patterns in the data alludes to the viability of machine learning models for predicting future gas prices.

6.2 Methodology

The gas price recommendation methodology we propose consists of two key components. First, we present a deep-learning based model to predict the gas price for a pre-defined period of time. Second, we introduce an algorithm that uses these predictions to recommend a gas price for a transaction, parameterised by the sender's willingness to delay the transaction. Both components, as well as the employed data pre-processing steps are presented in this section.

6.2.1 Gas Price Prediction

The methodology we propose requires a forecasting model to predict the gas price trajectory over a pre-defined number of time steps s. In particular, we are interested in predicting the minimum gas price under rational miner behaviour, since this can be seen as a lower bound for setting the gas price for a given transaction. From the preliminary data exploration in Section 6.1, it is apparent that the per-block data is extremely noisy, which can be attenuated by averaging over a longer period of time. We therefore average the minimum gas price of all blocks in consecutive 5 minute intervals and forecast on this level of granularity, instead of using per-block data directly. A time step is then defined as a 5 minute interval. We denote the complete time series of average minimum gas prices by y. Furthermore, we define the aggregated time series of all features used as model input as $\mathcal{D}$, where $d_t \in \mathcal{D}$ denotes the feature vector for a single time step t. For both model training and inference, we use a sliding window model that uses a fixed-size window of historical data with l time steps for prediction. The problem of forecasting a window of s time steps using a window of size l can then be defined as

$$\hat{y}_{t+1}, \ldots, \hat{y}_{t+s} = \underset{y_{t+1},\ldots,y_{t+s}}{\operatorname{argmax}} \; p(y_{t+1}, \ldots, y_{t+s} | d_{t-l}, ..., d_t) \,. \tag{6.1}$$

In the remainder of this section we present our pre-processing methodology and proposed forecasting model.

Feature Name	Lagged by 24h
Average gas price per block	Yes
Transaction count per block	No
Max. gas price per block	No
Min. gas price per block	No
ETH price at block timestamp	No

Table 6.2: Features used as input data for the predictive model to forecast the minimum gas price. Lagged variables are included both with and without lag.

Pre-processing

We introduce a number of pre-processing steps to the data, which specifically aim to reduce the impact of noise while still capturing seasonal components and trends. Table 6.2 lists the features used as input for the predictive model. Due to the daily seasonality in the data, some variables are also included with a lag of 24 hours. To reduce the impact of noise in the data, we first remove outliers using a heuristic criterion, where we delete all data points that are more than 1.5 standard deviations higher or lower than the mean. Subsequently, all data is normalised to values between 0 and 1. Since the main goal of the predictive model is to capture the seasonality and predict the gas price on a fairly coarse level, we employ a further pre-processing step presented in [PPL20]. This additional step applies a discrete Fourier transform to each window in the input data and truncates the frequency domain representation of the time series using an adaptive energy-based criterion. We then convert it back to the time-domain using an inverse Fourier transform. This methodology allows us to adaptively reduce the impact of short-term fluctuations in each window of input data, while still capturing the seasonal components and overall trend.

Model

As a forecasting model, we propose the use of a Gated Recurrent Unit (GRU) [CvMG+14]. GRUs are a specialisation of recurrent neural networks, where a computationally efficient gating mechanism is used. Gating has been shown to improve the network's ability to learn longer term dependencies [HS97], making this kind of model well-suited to the problem at hand. The

GRU architecture is given by

$$z_t = \sigma(W_z d_t + V_z h_{t-1} + b_z)\,, \tag{6.2}$$

$$r_t = \sigma(W_r d_t + V_r h_{t-1} + b_r)\,, \tag{6.3}$$

$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \phi(W_h d_t + V_h(r_t \circ h_{t-1}) + b_h)\,, \tag{6.4}$$

$$\hat{y} = \hat{y}_{t+1}, \ldots, \hat{y}_{t+s} = f(h_t)\,, \tag{6.5}$$

where $\circ$ denotes the Hadamard product, W, V and b are parameter matrices and biases, $\sigma(\cdot)$ and $\phi(\cdot)$ denote the sigmoid and hyperbolic tangent functions, respectively, z_t, r_t and h_t are the update and reset gates and the hidden state and $f(\cdot)$ denotes the final linear layer of the network. The network is trained using gradient descent and backpropagation with an Adam optimiser [KB14].

6.2.2 Recommendation Algorithm

We now describe our recommendation algorithm which leverages the gas prices predicted by our model. We use the 20th percentile of the predicted prices as the initial gas price, which we note $\hat{g}$. One of the main objectives of our algorithm is to scale $\hat{g}$ such that the faster the predicted gas prices are decreasing, the lower the gas price recommended by the algorithm. On the other hand, if the prices are increasing, the predicted prices should not be significantly lower than the current gas price. We incorporate this objective by finding a coefficient $0 < c \leq 1$ that is multiplied with the predicted gas price $\hat{g}$. Furthermore, we want c to increase or decrease exponentially with respect to the trend to achieve aggressive gas pricing if the predicted prices decrease quickly.

First, we compute the trend of the predictions $\hat{y}$ returned by our forecasting model. We fit a linear function such that $\hat{y} = aX + b$, with $X = 1, 2, \cdots, s$, and store the slope a, which captures the trend in the predicted gas prices. We then normalise a to $\tilde{a}$ to lie in the range between 0 to 1. This is achieved by computing the maximum A_{max} and minimum A_{min} values

of the slopes we obtain for our training data and computing $\tilde{a}$ according to Equation (6.6).

$$\tilde{a} = \frac{a - A_{min}}{A_{max} - A_{min}} \tag{6.6}$$

Then, to obtain the described exponential behaviour, we exploit the fact that the exponential function in the interval $[-2, 0]$ has the desired properties and hence, compute c using Equation (6.7).

$$c = e^{2\tilde{a}-2} \tag{6.7}$$

Finally, to allow the user to configure the urgency of a transaction, we define an urgency parameter $\mathcal{U}$, which we use to scale the obtained coefficient c to arrive at a recommended gas price $\mathcal{G}$ given by

$$\mathcal{G} = \hat{g} \cdot c \cdot \mathcal{U}\,. \tag{6.8}$$

Algorithm 1 Evaluation procedure of the gas recommendation efficiency

```
function EvaluateRecommender(StartBlock, EndBlock, Recommend)
    Pending ← ∅
    Results ← ∅
    Block ← StartBlock
    while Block ≤ EndBlock ∨ (Pending ≠ ∅ ∧ Block ≤ LastBlock) do
        Price ← GetMinimumPrice(Block)
        while Pending ≠ ∅ ∧ min_{t∈Pending} (t_1) ≥ Price do        ▷ t_1 is the transaction price
            Transaction ← argmin_{t∈Pending} (t_1)
            Pending ← Pending \ {Transaction}
            Results ← Results ∪ {(Transaction, Block, Price)}
        end while
        if Block ≤ EndBlock then
            Recommended ← Recommend(Block)
            Pending ← Pending ∪ {(Block, Recommended)}
        end if
        Block ← Block + 1
    end while
    return Results
end function
```

6.2.3 Measuring Gas Recommendation Efficiency

Up to here, we have described how we recommend a price at a given block number. However, to understand how optimal a gas price is, we need to measure the difference between the recommended and the optimal gas price.

To evaluate the efficiency of our approach, we iterate over a range of blocks, where we do the following. For each block, a new transaction using the recommended gas price is added to a set of pending transactions. Each transaction in the pending set is processed upon encountering a block with a minimum gas price lower than that specified in the transaction. We keep track of the recommended price, the inclusion price, i.e. the minimum gas price of the block where the transaction is included, and the number of blocks elapsed until inclusion. We show the detailed steps in Algorithm 1. The `EvaluateRecommender` function takes a start block, an end block and a recommendation function to evaluate. `LastBlock` is the number of the last block which we evaluate and `GetMinimumPrice` returns the minimum gas price for a given block.

To be able to evaluate the efficiency of our algorithm, we use the Geth gas price recommendation algorithm as the main baseline, as it is by far the most widely used Ethereum client [eth20c].

6.3 Results

In this section, we present the results we obtain when using the methodology presented in the previous section and compare them with our baselines.

6.3.1 Model Training

All models are implemented in Python, using the PyTorch library [PyT]. We train all models on a personal computer with 32GB of RAM, an 8^{th} generation Intel Core i7-8700 with 3.20GHz and 6 cores and a 256GB SATA hard drive. Model training and hyper parameter tuning is performed on the data between 10 November, 2019 to 20 November, 2019, where we use the

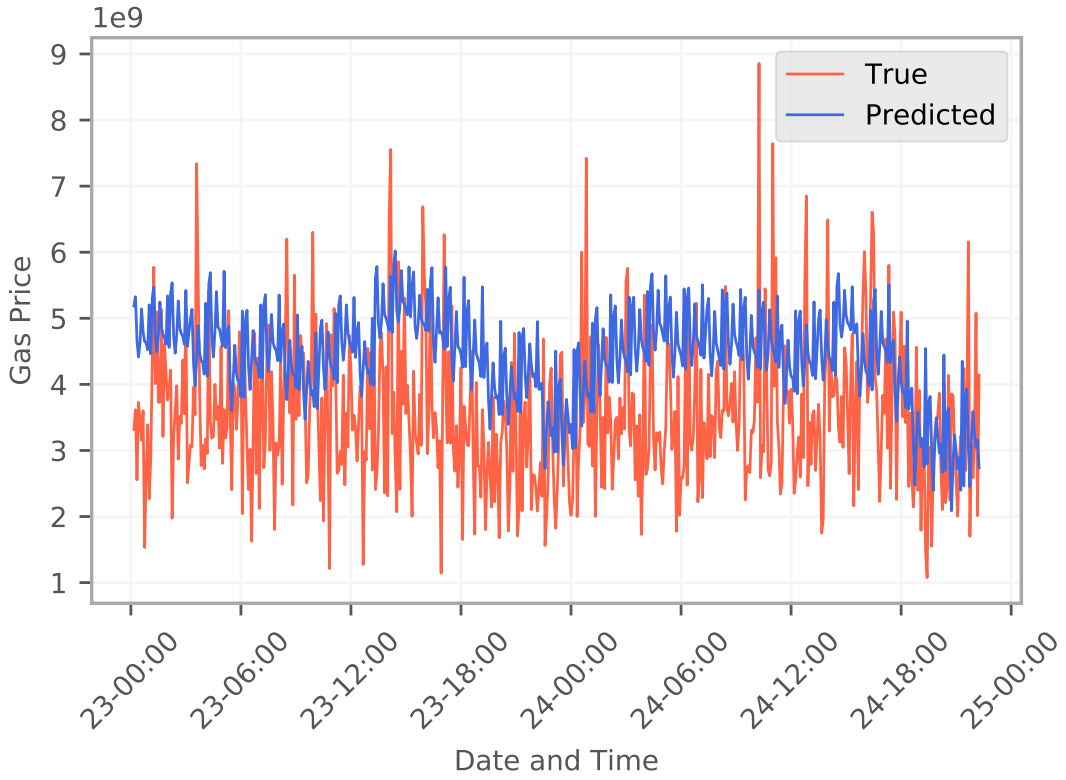

Figure 6.4: Exemplary gas price predictions obtained with our forecasting model for the period between the 23 November, 2019 and 25 November, 2019.

Model	Parameter	Description
Geth	Scaling ($\mathcal{S}$)	Ratio by which to scale the price (0.8 means use 80% of the recommended price)
proposed approach	Urgency ($\mathcal{U}$)	Urgency tuning parameter to trade-off price for time
Look-ahead	Blocks ($\mathcal{B}$)	Number of blocks to look ahead

Table 6.3: Parameters used in the different strategies

first 70% of the data for training and the remaining 30% for validation. We show exemplary predictions of our model in Figure 6.4.

6.3.2 Evaluation

We use a sample of around five days of data — from 20 November, 2019 (block 8,965,759) to November 24, 2019 (block 8,995,344) — and evaluate the different price recommendation strategies using the procedure presented in Algorithm 1. We first describe the parameters of each strategy in Table 6.3. For Geth we use a scaling ratio parameter $\mathcal{S}$ with which the

Strategy	Parameter	Gas price	Blocks waited
Geth	$\mathcal{S} = 1.0$	4,414,902,746	1.97
Geth	$\mathcal{S} = 0.9$	4,080,968,868	15.49
Geth	$\mathcal{S} = 0.8$	3,531,922,197	25.52
Look-ahead	$\mathcal{B} = 15$	1,166,965,099	4.80
Look-ahead	$\mathcal{B} = 30$	969,559,938	8.52
Look-ahead	$\mathcal{B} = 60$	782,105,012	18.84
Proposed approach	$\mathcal{U} = 1.0$	2,120,108,703	3.28
Proposed approach	$\mathcal{U} = 0.9$	1,908,097,833	3.79
Proposed approach	$\mathcal{U} = 0.8$	1,696,086,963	5.13
Proposed approach	$\mathcal{U} = 0.7$	1,484,076,092	10.06

Table 6.4: Results of the different recommendation strategies presented. Gas price and wait time are averaged over the number of blocks processed. Parameters are described in Table 6.3.

recommended gas price is multiplied. The main purpose of this parameter is to ensure that giving a lower gas price does have a direct impact on the number of blocks waited. Our proposed recommendation strategy accepts a single parameter $\mathcal{U}$ representing the urgency. The urgency parameter is used to trade off gas price for waiting time: the lower the urgency, the lower the gas price and hence, the longer the waiting time. Empirically, reasonable values for these parameters are roughly between 0.7 and 1.3, where 0.7 will result in cheap but long to be accepted transactions and 1.3 will result in more expensive but faster transactions. Finally, our look-ahead model, which we use to estimate the lowest possible price takes a parameter $\mathcal{B}$ representing the maximum look-ahead as a number of blocks. We note that the look-ahead strategy is for validation purposes only as it uses information about future blocks, which would obviously not be available in practice.

We present a summary of the results for the different recommendation strategies in Table 6.4. We use several values for the parameters of each strategy and order its results so that the gas price decreases and the number of blocks to wait increases. We can see that using the price recommended by Geth, the waiting time is very short — on average less than 2 blocks — with an average gas price of around 4.4 Gwei. However, by just using 90% of the recommended price, the waiting time increases to an average of 15.5 blocks. Comparing these results to the minimum possible gas price, obtained from the look-ahead model, we can see that by only waiting for an average of 4.8 blocks a saving of 75% could be obtained. Although these numbers

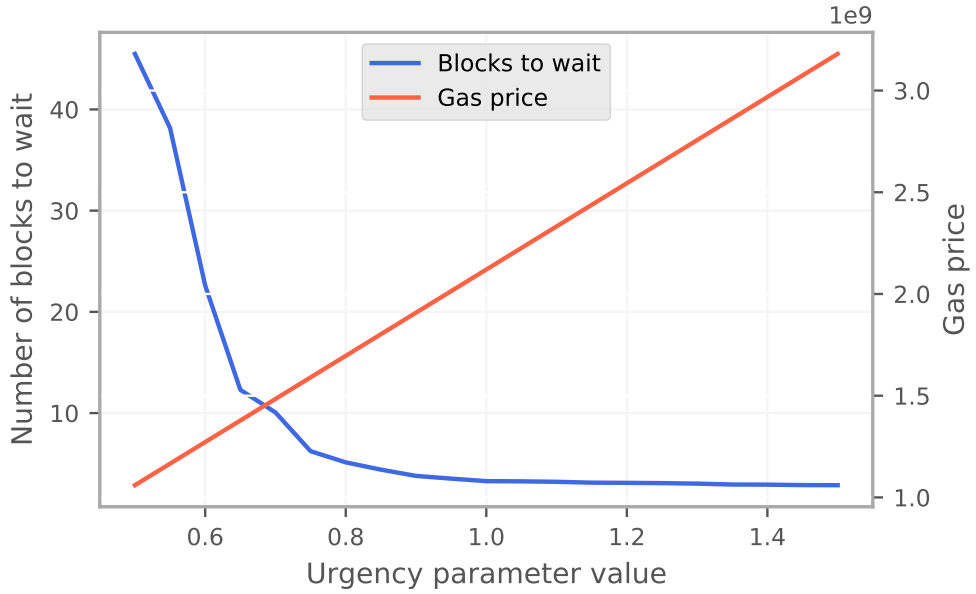

Figure 6.5: Effect of the urgency parameter on the average gas price paid and number of blocks waited.

are hypothetical, they suggest the potential for significant improvement.

We now show how our model performs in comparison to the price recommended by Geth and the hypothetical minimum price. With the urgency parameter set to 1.0, our model recommends a gas price on average twice as low as the Geth price, while waiting for an average of approximately 3.3 blocks. When decreasing the urgency parameter, we can see that the number of blocks elapsed increases fairly slowly at first but doubles between 0.8 and 0.7, showing that at this point the gas price becomes too low for the transaction to be included in a timely manner. In Figure 6.5, we show the effect of our urgency parameter on the average gas price paid and the average number of blocks elapsed until the transaction is included.

6.4 Related Work

For Ethereum in particular, there has been extensive research on smart contract correctness, upper-bound gas consumption and imperfections in the current EVM gas cost model. Nonetheless, very little work has been done with the goal of determining optimal gas prices. In this section, we first present existing work on the gas mechanism, before examining the most widely

used gas price recommendation methods.

6.4.1 Gas Mechanism

The overconsumption of gas can be harmful for the contract user for two main reasons: higher monetary costs and potential vulnerabilities. Gas overconsumption is examined by Chen et al. [CLLZ17], who focus on gas usage optimisation by introducing Gasper, a tool leveraging symbolic execution for detecting costly patterns in the bytecode of smart contracts which are not optimised by the Solidity compiler. Potential issues in the form of gas-related vulnerabilities are carefully examined by Grech et al. [GKJ+18], who propose a static analysis tool, called MadMax, predominantly suitable for detecting out-of-gas exceptions which may cause contract funds being locked. Elvira et al. [AGRS18] present Gastap, a static analysis tool for inferring gas upper bounds for smart contracts and are thereby able to detect whether any out-of-gas vulnerabilities could exist. A further approach for computing gas consumption upper bounds was introduced by Marescotti et al. [MBH+18], however, the authors are yet to implement and test their algorithms in an EVM setting. For a more general summary of existing smart contract verification tools we point the reader to [HK18].

There have been several pieces of work focusing on imperfections in the current gas cost mechanism. Both Yang et al. [YMRP19] and Perez and Livshits [PL19] identify inconsistencies in the pricing of EVM instructions in the current gas cost model. The latter propose a new type of attack aimed at exploiting EVM design flaws by generating resource exhaustive contracts, which are on average significantly slower in terms of throughput than typical contracts. As a means of preventing Denial-of-Service attacks stemming from under-priced EVM instructions a modification of the current gas cost mechanism has been proposed by [CLW+17].

While several pieces of existing work examine the current gas cost mechanism, limited work exists on gas price recommendation. Pierro et al. [PR19] investigate potential factors that influence transaction fees in Ethereum from a technical and economic perspective, yet leave a gas price prediction model for future research.

6.4.2 Gas Price Oracles

In the following, we examine existing approaches for gas price recommendation that are used in practice.

Geth. The Ethereum client implementation in go, namely Geth [Git20b], accounts for over 79% of all Ethereum clients [eth20c]. To recommend a gas price, Geth uses the minimum gas price of the previous blocks. It looks back at the 100 blocks preceding the current one and then uses the value of the 60th percentile of the minimum gas prices as the price recommendation.

EthGasStation. A further gas price oracle has been introduced by EthGasStation [eth20a], a third-party tool, which estimates the expected number of blocks required to confirm a transaction at a given gas price using a Poisson regression model based on data of the previous 10,000 blocks. This approach has also been implemented by the popular Ethereum block explorer Etherchain [eth20b]. Unfortunately, no historical data was available for comparison.

GasStation – Express. EthGasStation also released a more simple gas price oracle called "GasStation – Express" [Git20a]. This approach predicts the likelihood of a transaction being included in the next block at a given gas price by examining the percentage of the last 200 blocks that included a transaction with the same or lower gas price [Sta17]. The percentage thresholds of recent block inclusions are fixed for the categories *Fast* (90%), *Standard* (60%) and *SafeLow* (35%). Additionally a *Fastest* option is given, whereby the suggested gas price was included by all of the previous 200 mined blocks, which likely results in the sender overpaying considerably. Just like the threshold percentages, the associated expected confirmation times are also hard-coded, which limits the speed at which the system can react to changes.

6.4.3 Follow-up Work

In this section we discuss work that was released after our research presented in this chapter had been published. We believe that it is worth discussing some of these papers briefly, as we deem these relevant for the area of gas price prediction and stable transaction throughput.

Pierro and colleagues [PRTD20, PRD+22] conducted an investigation into the reliability of the ETH Gas Station and Ether Chain oracle. The ETH Gas Station asserts a 2% margin of error, yet their study reveals that the actual margin ranges from 4% to 28%, depending on the desired speed of the users.

Carl and Ewerhart [CE20] conducted a study on the seasonality of gas prices in Ethereum. They put forward a seasonal auto-regressive integrated moving average (SARIMA) model to forecast the hourly median of the gas price threshold.

In [LS21], the authors propose the QoS-driven Customizable Forecasting Framework (QCFF) as a link between decentralised applications and transaction fee (TF) prediction models. QCFF offers cost reduction for timely transaction confirmation in decentralised applications and provides a convenient framework for prediction model providers to integrate customised models. Future work involves designing personalised prediction models using machine learning or deep learning techniques tailored to specific decentralised applications' transaction data.

The authors of [MACK21] introduce a novel approach to predict Ethereum gas prices using machine learning models, specifically the Prophet algorithm, LSTM, and GRU. The performance of these models is evaluated and compared to the widely used gas price oracle, Geth. The results indicate that LSTM and GRU outperform both the Prophet model and Geth, exhibiting a lower mean squared error (MSE) of 0.008 compared to Geth's MSE of 0.016 and Prophet's MSE of 0.014. Overall, this study provides insights into the effectiveness of different prediction algorithms for Ethereum gas prices.

The goal of [LBF22] to minimise user expenses while ensuring a high probability of transaction processing. The authors introduce a novel method based on a Monte Carlo approach taking gaslimit and the pending list stateinto account to predict the probability of a transaction being mined within a specified time limit.

6.5 Conclusion

Motivated by an empirical analysis of 3 months of data, we have proposed a novel approach for recommending the Ethereum gas price that outperforms the method of the most widely used Ethereum client (at the time of research). Our approach uses a deep-learning based price forecasting model as well as an algorithm parameterised by an urgency value that can be set by the user. In a comprehensive evaluation, we show that our approach is able to reduce the average gas price paid by the sender of a transaction by more than 50% while only introducing an average additional waiting time of 1.3 blocks compared to Geth.

Our evaluation of the proposed approach aimed to focus on common-sized transactions. For more computationally intensive transactions, the gas price would likely need to be increased to ensure timely inclusion in a block. However, this could be easily accomplished by adjusting the urgency parameter.

Chapter 7

Conclusion

7.1 Summary of Thesis Achievements

This thesis explores the impact of misaligned incentive mechanisms on the stability of Proof-of-Work (PoW) blockchains, specifically focusing on block and transaction throughput. Unstable block throughput can lead to delays in transaction confirmations, resulting in higher fees, longer wait times, and decreased user satisfaction. It can also make the network vulnerable to attacks and compromise consensus. Therefore, stable block throughput is critical for the overall functionality and security of PoW blockchains. Internal and external incentive mechanisms in blockchain networks play a crucial role in shaping participant behaviour and affecting network security and efficiency. By understanding and aligning these incentives, more robust and stable networks can be created.

The extent to which these misalignments may result in attack vectors is evaluated, along with an assessment of past exploits. Furthermore, the thesis proposes more robust mechanisms that align incentives to ensure overall throughput stability and network security.

To support research on mining pools and reward payout schemes, an open-source simulation framework called PoolSim is presented. It enables the simulation of miner behaviour under various mining pool reward distribution schemes. The thesis extends existing research on queue-

based mining pool reward distribution manipulation strategies and evaluates their profitability. It also introduces the concept of "uncle traps", an attack specific to Ethereum queue-based mining pools, and provides empirical evidence of its impact on block throughput. We present strategies for increasing profits by employing uncle traps and subsequently propose a fix to the uncle block reward distribution mechanism that protects against the identified strategies.

The thesis further explores the design of internal stability mechanisms in PoW blockchains by focusing on the impact of difficulty adjustment algorithms on block throughput. Empirical evidence demonstrates that certain difficulty algorithms can lead to unstable block solve times, and miners' behaviour can contribute to this phenomenon through coin-hopping strategies. We present NEFDA, a difficulty algorithm based on a negative exponential filter to address this issue and ensure stable block solve times. The work on NEFDA was also cited by the official proposal to implement ASERT as the new BCH difficulty algorithm, which was activated on November 15, 2020.

Lastly, the thesis addresses transaction throughput improvement by presenting a novel gas price prediction model for Ethereum. The model combines a deep-learning-based price forecasting approach with a user-specific urgency value to recommend gas prices that balance timely inclusion and transaction cost. Empirical analysis, predictive modeling, and evaluation on real-world data demonstrate significant cost savings while incurring minimal delays compared to existing gas price recommendation mechanisms. The proposed model outperforms the most widely used gas price recommendation approach at the time of writing, resulting on average in costs savings of more than 50% while only incurring an inclusion delay of 1.3 blocks compared to Geth.

7.2 Future Work

7.2.1 Queue-based Mining Pool Reward Schemes

A further investigation into the overall fairness of queue-based mining pools including a game-theoretic evaluation of mining pool equilibria under different attack scenarios could be a fruitful

avenue for further research. However, this research would be less applicable in practice for Ethereum ever since Ethereum shifted to Proof-of-Stake and therefore queue-based mining pools are no longer being used. Nevertheless, a queue-based model could still be applied to mining pools that mine on PoW chains. Depending on the underlying PoW chain, a formal exploration of schemes to mitigate the deliberate increase in a mining pool's uncle/orphan rate could further help devise fair reward distribution policies. Lastly, formal verification of mining pool reward schemes would help to increase transparency and guarantee fairness with respect to reward allocation.

7.2.2 Extending PoolSim

We believe that *PoolSim* can facilitate research on areas of mining pool reward schemes, such as fairness, vulnerabilities and attacks. As there has been the least amount of academic research on the queue-based reward scheme, we shift our focus to this scheme when pointing towards areas of future research in the context of *PoolSim*. From the few simulations examined in this thesis, we were able to make some interesting observations. We showed that despite working in a two miner scenario, queue-based attack strategies are not necessarily nearly as effective in a multi-miner pool. This is presumably caused by the pool size and the hash rate distribution of the pool, as these variables directly affect the credit differences between miners in the pool.

Even though pool-hopping on a luck basis between two queue-based pools did not provide any novel insights, we were able to successfully demonstrate the powerful conditional pool-hopping functionality of *PoolSim*. A feature, which, to the best of our knowledge, has not been implemented and utilised elsewhere. Hence, with regards to employing *PoolSim* for future research, one could with very little effort construct a conditional pool-hopping scenario between multiple queue-based mining pools and add a more complex condition for submitting shares. An example of such a condition would be to find the optimal pool out of a set of queue-based mining pools in terms of highest average proportion of credits lost per round. Furthermore, this conditional logic could be extended to also employ a strategy such as the tactical donation of shares in order to actually exploit large credit differences when they occur. Such an exploratory

approach could provide further insights into hash rate distributions of queue-based mining pools and their implications for the effectiveness of different attacks targeting the reset mechanism.

A additional area of future interest could lie in the game-theoretic aspects and analysis of multi-attacker scenarios in different pool constellations. Even though we only used the framework to look into single-attacker scenarios in two and multi-miner settings, *PoolSim* can be used to simulate multi-attacker scenarios. This could provide stimulating insights into examining the effects of mining scenarios in large pools, where multiple (or perhaps only) attackers exist, all pursuing the same or different attack strategies.

7.2.3 Extensions to gas price prediction model

Future work can examine the usefulness of additional data, such as memory pool data, as model inputs. Additionally, the evaluation and comparison of our approach and previous approaches in a larger simulation may be a fruitful avenue for further research.

It should be noted that after this research was published, Ethereum switched the fee market design of the original Ethereum chain in a proposal called EIP-1559 [BCD+19], which is briefly explained in the next sentences. The update has a fixed charge for gas usage, which can fluctuate in each block based on a formula that considers the gas used in the previous block and the target gas limit of that block divided by an elasticity multiplier. The formula causes the fixed charge per unit of gas to increase when blocks exceed the gas target and decrease when blocks fall below the gas target. This fixed charge per gas is eliminated from circulation.

When making transactions, users specify the highest fee per gas they are willing to offer to miners as an incentive for including their transaction (referred to as the priority fee). Transactions also indicate the maximum fee per gas they are willing to pay overall (known as the maximum fee), which covers both the priority fee and the base fee per gas for the block. The transaction will always pay the base fee per gas of the block in which it is included and will pay the priority fee per gas specified in the transaction, as long as the combined amount of these two fees does not exceed the transaction's maximum fee per gas. Therefore, the presented research on gas

price prediction only applies to the Ethereum gas model pre-EIP-1559. Nonetheless, it would be interesting to examine the existing empirical gas price data under EIP-1559 and see whether any forms of cyclicality exist and how user behaviour has changed pre- and post-EIP-1559 with respect to setting gas prices.

Appendices

Appendix A

Address	Miner
qpk4hk3wuxe2uqtqc97n8atzrrr6r5mleczf9sur4h	BTC.TOP
qqfc3lxxylme0w87c5j2wdmsqln6e844xcmsdssvzy	Antpool
qrcuqadqrzp2uztjl9wn5sthepkg22majyxw4gmv6p	ViaBTC
qrd9khmeg4nqag3h5gzu8vjt537pm7le85lcauzezc	BTC.com
qrjc9yecwkldlhzys3euqz68f78s2wjxw5h6j9rqpq	Huobi Pool
qp4ajqctqvx5m5fhpswdkgm9whwsapgst5twl9zd5h	unknown
qqq9v3hhl0vga8w5cts6dx5aa8xep2v2ssvppp5xcn	unknown
qzkuv6ftvt28v6hauv44r58tjupsgn3nqsnslfxzqf	unknown

Table A.1: The Bitcoin Cash addresses for selected miners.

Bibliography

[AAV20] AAVE. AAVE. https://aave.com/, 2020. Accessed: 2 June 2023.

[AGRS18] Elvira Albert, Pablo Gordillo, Albert Rubio, and Ilya Sergey. GASTAP: A Gas Analyzer for Smart Contracts. *CoRR*, abs/1811.1, nov 2018.

[Ant23] AntPool. AntPool. https://www.antpool.com/home, 2023. Accessed: 2023-06-14.

[AT19] Vipul Aggarwal and Yong Tan. A Structural Analysis of Bitcoin Cash's Emergency Difficulty Adjustment Algorithm. *Available at SSRN 3383739*, 2019.

[BCD+19] Vitalik Buterin, Eric Conner, Rick Dudley, Matthew Slipper, Ian Norden, and Abdelhamid Bakhta. EIP-1559: Fee market change for ETH 1.0 chain, April 2019.

[BG17] Vitalik Buterin and Virgil Griffith. Casper the Friendly Finality Gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[Bit17] BitcoinABC. Difficulty Adjustment Algorithm Update. https://www.bitcoinabc.org/2017-11-01-DAA, 2017. Accessed: 2019-10-16.

[Bit18] BitcoinTalk. Don't be a fool - Stop mining on Ethpool right now. https://bitcointalk.org/index.php?topic=3970569.0, 2018. Accessed: 2018-09-17.

[Bit20] BitcoinABC. Bitcoin ABC. https://www.bitcoinabc.org, 2020. Accessed: 2020-03-02.

[Bit23] BitcoinCash. Bitcoincash, 2023. Accessed: 2023-06-05.

[BL17] George Bissias and Brian Neil Levine. Bobtail: A Proof-of-Work Target that Minimizes Blockchain Mining Variance (Draft), 2017.

[Bov11] Alex Boverman. Timejacking & Bitcoin. `https://culubas.blogspot.com/2011/05/timejacking-bitcoin_802.html`, 2011. Accessed: 2019-11-15.

[BTL19] George Bissias, David Thibodeau, and Brian N Levine. Bonded Mining: Difficulty Adjustment by Miner Commitment. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 372–390. Springer, 2019.

[But] Vitalik Buterin. EIP 150: Gas cost changes for IO-heavy operations . `https://eips.ethereum.org/EIPS/eip-150`. [Online; accessed 05-June-2019].

[But14] Vitalik Buterin. Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform. `https://github.com/ethereum/wiki/wiki/White-Paper`, 2014. Accessed: 2016-08-22.

[CB14] Nicolas T Courtois and Lear Bahack. On Subversive Miner Strategies and Block Withholding Attack in Bitcoin Digital Currency. *arXiv preprint arXiv:1402.1718*, 2014.

[CDE+16] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On scaling decentralized blockchains. In *International Conference on Financial Cryptography and Data Security*, volume 9604, pages 106–125, Berlin, Heidelberg, 2016. Springer.

[CE20] David Carl and Christian Ewerhart. Ethereum Gas Price Statistics. *University of Zurich, Department of Economics, Working Paper*, (373), 2020.

[Cir20] Circle. USDC. `https://www.circle.com/en/usdc`, 2020. Accessed: 2 June 2023.

[CLLZ17] Ting Chen, Xiaoqi Li, Xiapu Luo, and Xiaosong Zhang. Under-optimized smart contracts devour your money. *SANER 2017 - 24th IEEE International Conference on Software Analysis, Evolution, and Reengineering*, pages 442–446, 2017.

[CLW+17] Ting Chen, Xiaoqi Li, Ying Wang, Jiachi Chen, Zihao Li, Xiapu Luo, Man Ho Au, and Xiaosong Zhang. An Adaptive Gas Cost Mechanism for Ethereum to Defend Against Under-Priced DoS Attacks. In Joseph K. Liu and Pierangela Samarati, editors, *Information Security Practice and Experience*, pages 3–24, Cham, 2017. Springer International Publishing.

[com18] Ethereum community. Mining. `https://github.com/ethereum/wiki/wiki/Mining`, 2018. Accessed: 2018-09-13.

[Com19] Compound. Compound Finance. `https://compound.finance/`, 2019. Accessed: 2022-06-04.

[CvMG+14] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *CoRR*, abs/1406.1078, 2014.

[Dan17] Chris Dannen. *Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners*. Apress, Berkely, CA, USA, 1st edition, 2017.

[DW13] Christian Decker and Roger Wattenhofer. Information propagation in the Bitcoin network. In *IEEE P2P 2013 Proceedings*, pages 1–10, Trento, Italy, Sept 2013. IEEE Computer Society.

[DYQ+22] Yuhang Ding, Zihan Yang, Bo Qin, Qin Wang, Yanran Zhang, and Qianhong Wu. Squeezing Network Performance for Secure and Efficient PoW with Better Difficulty Adjustment. In *2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 874–881. IEEE, 2022.

[DZ21] Lu Ding and Yong Zhao. Uncle Incentive Mechanism Adjustment against Selfish Mining in Ethereum. In *2021 7th International Conference on Computer and Communications (ICCC)*, pages 1570–1574. IEEE, 2021.

[Ego19] Michael Egorov. Stableswap - efficient mechanism for Stablecoin liquidity, 2019.

[Eth] Etherscan. Ethereum Average Block Time Chart. https://etherscan.io/chart/blocktime. Accessed 28-January-2020.

[eth20a] EthGasStation. `https://ethgasstation.info/`, 2020. "[Online; accessed 14-January-2020]".

[eth20b] etherchain.org. Gas Price Oracle. `https://www.etherchain.org/tools/gasPriceOracle`, 2020. [Online; accessed 15-January-2020].

[eth20c] ethernodes.org. Ethereum Mainnet Statistics. `https://www.ethernodes.org/`, 2020. "[Online; accessed 15-January-2020]".

[FM18] Daniel Fullmer and A Stephen Morse. Analysis of Difficulty Control in Bitcoin and Proof-of-Work Blockchains. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 5988–5992. IEEE, 2018.

[for] forklol. `https://fork.lol/reward/dari/btc`. Accessed: 2020-03-31.

[fTCL20] freetrader, Jonathan Toomim, Calin Culianu, and Mark Lundeberg. 2020-nov-15 ASERT difficulty adjustment algorithm (aserti3-2d), 2020. Accessed: 13-11-2020.

[Git20a] Github. Gasstation-express. `https://github.com/ethgasstation/gasstation-express-oracle`, 2020. [Online; accessed 15-January-2020].

[Git20b] Github. Official Go implementation of the Ethereum protocol. `https://github.com/ethereum/go-ethereum/`, 2020. [Online; accessed 14-January-2020].

[Git20c] Github. Parity Ethereum: The Fastest and most Advanced Ethereum Client. `https://github.com/paritytech/parity-ethereum`, 2020. [Online; accessed 16-January-2020].

[GKJ+18] Neville Grech, Michael Kong, Anton Jurisevic, Lexi Brent, Bernhard Scholz, and Yannis Smaragdakis. MadMax: Surviving Out-of-Gas Conditions in Ethereum Smart Contracts. *SPLASH 2018 Oopsla*, 2(October), 2018.

[GKZ19] Peter Gaži, Aggelos Kiayias, and Dionysis Zindros. Proof-of-stake Sidechains. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 139–156. IEEE, 2019.

[GMSR+19] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. SoK: Off The Chain Transactions. Cryptology ePrint Archive, Report 2019/360, 2019. `https://eprint.iacr.org/2019/360`.

[Gol23] Bitcoin Gold. Bitcoin Gold. `https://bitcoingold.org/`, 2023. Accessed: 5 June 2023.

[GWPK20] Lewis Gudgeon, Sam Werner, Daniel Perez, and William J Knottenbelt. DeFi Protocols for Loanable Funds: Interest rates, Liquidity and Market Efficiency. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 92–112, 2020.

[HCD+18] Jordan Holland, Joseph Connor, Parker Diamond, Jared M. Smith, and Max. Schuchard. Not So Predictable Mining Pools: Attacking Solo Mining Pools by Bagging Blocks and Conning Competitors. `https://fc18.ifca.ai/preproceedings/79.pdf`, 2018. Accessed: 2018-10-19.

[HJNARP+19] Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, Alejandro Ranchal-Pedrosa, Cristina Pérez-Solà, and Joaquin Garcia-Alfaro. On the difficulty of hiding the balance of lightning network channels. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, pages 602–612, 2019.

[HK17] Geir Hovland and Jan Kucera. Nonlinear Feedback Control and Stability Analysis of a Proof-of-Work Blockchain. *Modeling, Identification and Control*, 38(4):157–168, 2017.

[HK18] Dominik Harz and William Knottenbelt. Towards Safer Smart Contracts: A Survey of Languages and Verification Methods, 2018.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[Hue23] Samuel Huestis. Cryptocurrency's Energy Consumption Problem. *Rocky Mountain Institute*, 2023. Accessed: 14 June, 2023.

[IWSK21] Dragos I Ilie, Sam M Werner, Iain D Stewart, and William J Knottenbelt. Unstable Throughput: When the Difficulty Algorithm Breaks. In *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–5. IEEE, 2021.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KKSK19] Yujin Kwon, Hyoungshick Kim, Jinwoo Shin, and Yongdae Kim. Bitcoin vs. Bitcoin Cash: Coexistence or Downfall of Bitcoin Cash? *arXiv preprint arXiv:1902.11064*, 2019.

[KL18] Tamás Király and Lilla Lomoschitz. Profitability of the coin-hopping strategy. *EGRES quick proof*, (2018-03), 2018.

[KMZ20] Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. Non-Interactive Proofs of Proof-of-Work. In *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers 24*, pages 505–522. Springer, 2020.

[Kra16] Daniel Kraft. Difficulty Control for Blockchain-Based Consensus Systems. In *Peer-to-Peer Networking and Applications*, volume 9, pages 397–413, 2016.

[KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In *Advances*

in Cryptology–CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part I, pages 357–388. Springer, 2017.

[LBF22] Arnaud Laurent, Luce Brotcorne, and Bernard Fortz. Transaction fees optimization in the Ethereum blockchain. *Blockchain: Research and Applications*, 3(3):100074, 2022.

[LJG15] Aron Laszka, Benjamin Johnson, and Jens Grossklags. When Bitcoin Mining Pools Run Dry: A Game-Theoretic Analysis of the Long-Term Impact of Attacks Between Mining Pools. In Michael Brenner, Nicolas Christin, Benjamin Johnson, and Kurt Rohloff, editors, *Financial Cryptography and Data Security*, pages 63–77, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[LNZ+16] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A Secure Sharding Protocol For Open Blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 17–30, 2016.

[LS21] Chia-Chun Lien and Wei-Tsung Su. A QoS-driven Customizable Forecasting Framework for Blockchain Transaction Fee Recommendation. In *2021 IEEE International Conference on Smart Internet of Things (SmartIoT)*, pages 348–349. IEEE, 2021.

[LSP+15] Loi Luu, Ratul Saha, Inian Parameshwaran, Prateek Saxena, and Aquinas Hobor. On Power Splitting Games in Distributed Computation: The Case of Bitcoin Pooled Mining. In *Proc. 28th IEEE Computer Security Foundations Symposium (CSF 2015)*, pages 397–411, Verona, Italy, 2015. IEEE, IEEE Computer Society.

[LSZ15] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. Inclusive Block Chain Protocols. In *Financial Cryptography and Data Security: 19th International*

Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers 19, pages 528–547. Springer, 2015.

[Lun19] Mark B. Lundeberg. Static difficulty adjustments, with absolutely scheduled exponentially rising targets (DA-ASERT). unpublished, 2019.

[Lun20] Mark B. Lundeberg. Static difficulty adjustments, with absolutely scheduled exponentially rising targets (DA-ASERT) — v.2. Accessed: 2023-06-15, 2020.

[MACK21] Rawya Mars, Amal Abid, Saoussen Cheikhrouhou, and Slim Kallel. A Machine Learning Approach for Gas Price Prediction in Ethereum Blockchain. In *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 156–165. IEEE, 2021.

[Mak19] MakerDAO. MakerDAO. `https://makerdao.com/en/`, 2019. Accessed: 24 July 2020.

[MBH+18] Matteo Marescotti, Martin Blicha, Antti E J Hyvärinen, Sepideh Asadi, and Natasha Sharygina. Computing Exact Worst-Case Gas Consumption for Smart Contracts. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice*, pages 450–465, Cham, 2018. Springer International Publishing.

[MCJ17] Dmitry Meshkov, Alexander Chepurnoy, and Marc Jansen. Short paper: Revisiting difficulty control for blockchain systems. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 429–436. Springer, 2017.

[Nak08] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. `https://bitcoin.org/bitcoin.pdf`, Dec 2008. Accessed: 2015-07-01.

[PD16] Joseph Poon and Thaddeus Dryja. The Bitcoin Lightning Network: Off-Chain Instant Payments, 2016.

[PL19] Daniel Perez and Benjamin Livshits. Broken Metre: Attacking Resource Metering in EVM. *arXiv preprint arXiv:1909.07220*, 2019.

[PPL20] Paul J. Pritz, Daniel Perez, and Kin K. Leung. Fast-Fourier-Forecasting Resource Utilisation in Distributed Systems. In *29th International Conference on Computer Communications and Networks, ICCCN 2020, Honolulu, HI, USA, August 3-6, 2020*, pages 1–9. IEEE, 2020.

[PR19] Giuseppe Antonio Pierro and Henrique Rocha. The Influence Factors on Ethereum Transaction Fees. In *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, pages 24–31. IEEE, 2019.

[PRD+22] Giuseppe Antonio Pierro, Henrique Rocha, Stéphane Ducasse, Michele Marchesi, and Roberto Tonelli. A User-Oriented Model for Oracles' Gas Price Prediction. *Future Generation Computer Systems*, 128:142–157, 2022.

[PRTD20] Giuseppe Antonio Pierro, Henrique Rocha, Roberto Tonelli, and Stéphane Ducasse. Are the Gas Prices Oracle Reliable? A Case Study using the EthGasStation. In *2020 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pages 1–8. IEEE, 2020.

[PvW08] Wouter Penard and Tim van Werkhoven. On the Secure Hash Algorithm family. *Cryptography in Context*, pages 1–18, 2008.

[PWXL21] Daniel Perez, Sam M Werner, Jiahua Xu, and Benjamin Livshits. Liquidations: DeFi on a Knife-edge. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part II 25*, pages 457–476. Springer, 2021.

[PyT] PyTorch Contributors. PyTorch Online Documentation. https://pytorch.org/docs. Accessed 13-August-2019.

[PZS+16] Pavel Prihodko, Slava Zhigulin, Mykola Sahno, Aleksei Ostrovskiy, and Olaoluwa Osuntokun. Flare: An Approach to Routing in Lightning Network. *White Paper*, 144, 2016.

[Res19] BitMEX Research. Bitcoin Cash's October 2019 Hashrate Volatility Increase. https://blog.bitmex.com/bitcoin-cashs-october-2019-hashrate-volatility-increase/, 2019. Accessed: 2019-11-08.

[Ros11] Meni Rosenfeld. Analysis of Bitcoin Pooled Mining Reward Systems. arXiv preprint:1112.4980, 2011. Accessed:2018-10-16.

[Sec20] Amaury Sechet. Announcing the Grasberg DAA, 2020. Accessed: 13-11-2020.

[Sta17] Medium: ETH Gas Station. GasStation Express: A simple gas price oracle for anyone running a full Ethereum node. https://medium.com/@ethgasstation/gasstation-express-a-simple-gas-price-oracle-for-anyone-running-a-full-ethereum-node-f1bde46260f5, 2017. [Online; accessed 15-January-2020].

[SV23] Bitcoin SV. Bitcoin SV. https://bitcoinsv.io/, 2023. Accessed: 2019-10-21.

[Sza18] Pawel Szalachowski. (short Paper) Towards More Reliable Bitcoin Timestamps. In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 101–104. IEEE, 2018.

[Tay17] Michael Bedford Taylor. The Evolution of Bitcoin Hardware. *Computer*, 50(9):58–66, 2017.

[Too20] Jonathan Toomim. The BCH difficulty adjustment algorithm is broken. here's how to fix it. https://www.reddit.com/r/btc/comments/fanc6o/the_bch_difficulty_adjustment_algorithm_is_broken/, 2020. Accessed: 2020-03-01.

[Uni20] Uniswap. Uniswap whitepaper, 2020. Accessed: 26-08-2020.

[VTM14] Marie Vasek, Micah Thornton, and Tyler Moore. Empirical analysis of denial-of-service attacks in the Bitcoin ecosystem. In *International Conference on Financial Cryptography and Data Security*, pages 57–71, Berlin, Heidelberg, 2014. Springer, Springer Berlin Heidelberg.

[Woo14] Gavin Wood. Ethereum: A Secure Decentralised Generalised Transaction Ledger. `http://bitcoinaffiliatelist.com/wp-content/uploads/ethereum.pdf`, 2014. Accessed: 2016-08-22.

[WP20] Sam M Werner and Daniel Perez. PoolSim: A Discrete-Event Mining Pool Simulation Framework. In *Mathematical Research for Blockchain Economy: 1st International Conference MARBLE 2019, Santorini, Greece*, pages 167–182. Springer, 2020.

[WPG+22] Sam M Werner, Daniel Perez, Lewis Gudgeon, Ariah Klages-Mundt, Dominik Harz, and William J Knottenbelt. SoK: Decentralized Finance (DeFi). In *AFT '22: Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, pages 30–46. ACM, 2022.

[WPP20] Sam M Werner, Paul J Pritz, and Daniel Perez. Step on the Gas? A Better Approach for Recommending the Ethereum Gas Price. In *Mathematical Research for Blockchain Economy: 2nd International Conference MARBLE 2020, Vilamoura, Portugal*, pages 161–177. Springer, 2020.

[WPZK19] Sam M Werner, Paul J Pritz, Alexei Zamyatin, and William J Knottenbelt. Uncle Traps: Harvesting Rewards in a Queue-based Ethereum Mining Pool. In *Proceedings of the 12th EAI International Conference on Performance Evaluation Methodologies and Tools*, pages 127–134, 2019.

[WSNH19] Gang Wang, Zhijie Jerry Shi, Mark Nixon, and Song Han. SoK: Sharding on Blockchain. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 41–61, 2019.

[YMRP19] Renlord Yang, Toby Murray, Paul Rimba, and Udaya Parampalli. Empirically Analyzing Ethereum's Gas Mechanism. *CoRR*, abs/1905.0, 2019.

[YWY+20] Guangsheng Yu, Xu Wang, Kan Yu, Wei Ni, J Andrew Zhang, and Ren Ping Liu. Survey: Sharding in blockchains. *IEEE Access*, 8:14155–14181, 2020.

[zaw18] zawy12. Timestamp Attacks. https://github.com/zawy12/difficulty-algorithms/issues/30, 2018. Accessed: 2019-11-15.

[zaw19a] zawy12. BCH needs a new DA. https://github.com/zawy12/difficulty-algorithms/issues/47, 2019. Accessed: 2019-10-01.

[zaw19b] zawy12. Oscillations in Simple Moving Averages. https://github.com/zawy12/difficulty-algorithms/issues/48, 2019. Accessed: 2019-10-01.

[zaw19c] zawy12. Summary of Difficulty Algorithms. https://github.com/zawy12/difficulty-algorithms/issues/50, 2019. Accessed: 2020-03-02.

[zaw19d] zawy12. Using EMA for BCH's new DA. https://github.com/zawy12/difficulty-algorithms/issues/49, 2019. Accessed: 2020-03-02.

[ZSJ+19] Alexei Zamyatin, Nicholas Stifter, Aljosha Judmayer, Philipp Schindler, Edgar Weippl, and William J Knottenbelt. A Wild Velvet Fork Appears! Inclusive Blockchain Protocol Changes in Practice: (Short Paper). In *Financial Cryptography and Data Security: FC 2018 International Workshops, BITCOIN, VOTING, and WTSC, Nieuwpoort, Curaçao, March 2, 2018, Revised Selected Papers 22*, pages 31–42. Springer, 2019.

[ZWW+17] Alexei Zamyatin, Katinka Wolter, Sam Werner, Peter G Harrison, Catherine EA Mulligan, and William J Knottenbelt. Swimming with Fishes and Sharks: Beneath the Surface of Queue-based Ethereum Mining Pools. In *2017 IEEE 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 99–109, Banff, Canada, Sept 2017. IEEE, IEEE Computing Society.

[ZXE+22] Liyi Zhou, Xihan Xiong, Jens Ernstberger, Stefanos Chaliasos, Zhipeng Wang, Ye Wang, Kaihua Qin, Roger Wattenhofer, Dawn Song, and Arthur Gervais. SoK: Decentralized Finance (DeFi) Attacks. *Cryptology ePrint Archive*, 2022.

[ZZW+20] Ren Zhang, Dingwei Zhang, Quake Wang, Shichen Wu, Jan Xie, and Bart Preneel. NC-Max: Breaking the Security-Performance Tradeoff in Nakamoto

Consensus. Cryptology ePrint Archive, Paper 2020/1101, 2020. https://eprint.iacr.org/2020/1101.