

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

After You: Intelligent Orchestration of Queue-based Cryptocurrency Mining Pools

Author:
Sam M. Werner

Supervisor:
Prof. William J. Knottenbelt

Submitted in partial fulfillment of the requirements for the MSc degree in MSc Computing
Science of Imperial College London

September 2017

Abstract

Cryptocurrency mining has been seen as a means to turn electricity into digital gold. It involves the guessing of a solution to a cryptographic puzzle, where the difficulty is decided upon by the network. Individual miners may increase their aggregate chance of finding a valid solution to the puzzle by joining a mining pool and subsequently combine their computational efforts. Mining pools employ different reward schemes in order to fairly distribute block rewards amongst miners and have thus become a favorable means for individual miners to receive steadier payout streams. A new type of reward scheme has been introduced by Ethpool, which uses a queue-based mechanism for distributing rewards amongst its miners.

This dissertation examines the workings of a queue-based mining pool, as implemented by Ethpool, and thoroughly assesses the effectiveness of potential mining strategies which may be employed in such a scheme. Hence a simulation framework was built in order to allow for the modeling of queue-based mining pools. In addition, mining strategies which aim at exploiting a mechanism specific to queue-based mining pools have been proposed. In order to determine the effectiveness of these strategies, simulations of two miner and multi-miner pools have been conducted and the strategies' performance assessed.

The main findings arising from this dissertation suggest that miners with above average hash rates can benefit from strategic mining in queue-based pools, given that certain conditions prevail. The nature of these conditions is rooted in the hash rate distribution of a queue-based pool and has thus made way for promising areas of future research.

Acknowledgments

I would like to thank the following people, without whom this MSc dissertation would never have been possible:

- My supervisor, Prof. William J. Knottenbelt, for his inspiration, encouragement and guidance, as well as for teaching me about the beauty of C++ programming and introducing me to the exciting world of cryptocurrencies.
- My family for their ongoing love, belief and support.
- My short term research colleague and friend, Alexei Zamyatin, for his patience and support in helping me familiarize myself with queue-based reward payout schemes.
- Paul Pritz and Fayzal Ghantiwala, two great friends, for insightful conversations and discussions regarding mining strategies.
- My friends for being encouraging and supportive at all times.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Aims and objectives	2
1.3	Contributions	2
1.3.1	A simulation framework for queue-based cryptocurrency mining pools	2
1.3.2	Potential issues with a queue-based payout scheme	3
1.3.3	A comparison of mining strategies in queue-based mining pools	3
1.3.4	Limitations of mining strategies in queue-based mining pools	3
1.4	Outline	4
1.5	Statement of originality and publications	5
2	Cryptocurrency mining, pools and reward schemes	6
2.1	Cryptocurrency mining	6
2.2	Solo mining	8
2.3	Mining pools	9
2.4	Miner rewards and reward payout schemes	9
3	Queue-based mining pools: Ethpool	12
3.1	Manipulating the queue: strategic mining	13
3.2	Share withholding	14
3.3	Tactical donation of mining power	15
3.4	Second payout address (wallet)	16
3.5	Pool hopping	17
4	EthSim: a mining pool simulator	18
4.1	The aims for the simulator	18
4.2	Design	19
4.2.1	Simulating a queue-based mining pool	19
4.2.2	Simulating mining strategies	19
4.2.3	System design	21
4.3	Implementation: a queue-based mining pool	22
4.3.1	Event queue and events	22
4.3.2	Share scheduling	22
4.4	Implementation: mining strategies (scenarios)	24
4.4.1	Conditions and actions	24
4.4.2	Scenario set up: condition and action	25
4.5	Setting up a simulation	27
4.5.1	The simulation configuration and data collection	28

4.5.2	Populating the pool	29
5	The <i>Two Miner Case</i>	30
5.1	A simple scenario	30
5.2	Simulation configuration	30
5.3	Simulation results and evaluation	31
5.3.1	Simulation of a two miner pool	31
5.3.2	Problems with the two miner case	35
6	The <i>Multi-Miner Case</i>	36
6.1	Simulation configuration	36
6.2	Naive conditions in the multi-miner case	37
6.3	Expectation-based conditions	41
6.3.1	Expected end of round queue constellation	43
6.3.2	Expectation-based condition 1: <i>ExpVal1</i>	43
6.3.3	ExpVal1: simulation results	45
6.3.4	Expectation-based condition 2: <i>ExpVal2</i>	45
6.3.5	ExpVal2: a hypothetical example	47
6.3.6	ExpVal2: simulation results	49
6.4	Luck-based condition	50
6.4.1	Hopping: a theoretical approach	50
6.4.2	Simulation set up	50
6.4.3	Hopping: simulation results	50
6.5	Exponential difficulty	51
6.6	Evaluation of queue-based mining strategies	52
7	Conclusion	58
7.1	Summary of Achievements	58
7.2	Applications	59
7.3	Future Work	59
7.3.1	Formal methods	59
7.3.2	An in-depth pool-hopping analysis	59
7.3.3	New mining strategies	60
	Glossary	63
	Appendices	66
A		67
A.1	Ethpool	67
A.2	Hash rate distributions	67
B		69
B.1	Design: EthSim	69

List of Figures

2.1	A representation of a blockchain and the relationship between individual blocks [7].	6
2.2	Screenshot: the top ten cryptocurrencies by market capitalisation	8
3.1	Screenshot: the maximum priority queue as implemented by Ethpool showing the fifteen miners with the highest accumulated credit balances.	12
3.2	Screenshot: A miner in Ethpool experiences a sudden increase in his hash rate for a short period of time [20].	16
4.1	A visualisation of the share scheduling and execution mechanism.	20
4.2	A UML diagram showing the relationships between the different classes of EthSim.	23
4.3	A UML diagram showing the relationship of the different event classes and event queue	24
4.4	A UML diagram exclusively showing the relationship between the different action and condition classes	27
5.1	The credit development of two miners for the first 100 blocks	32
5.2	The number of hashes computed by two miners per rewarded block correlated with pool luck for the first 100 blocks	33
5.3	The credit development for two miners in a share withholding attack scenario for the first 100 blocks	34
6.1	The distribution of hash rates of 729 Ethpool miners (logarithmic scale). [20]	37
6.2	The distribution of hash rates of 1,000 miners in the constructed mining pool	38
6.3	Development of the top miners' credits when winning a block compared to their performed work per block, as well as the ratio of the winning miner's credits to his performed number of hashes per block in a pool of 1 000 miners for 200 000 blocks.	39
6.4	The ratio of performed work per block in Ethpool relative to the average amount of performed work per block (miners have been grouped by their hash rate).[20]	40
6.5	Screenshot: Ethereum block difficulty growth chart [2]	51
6.6	The distribution of mining power in a constructed pool of 10 miners (logarithmic scale).	53
6.7	The distribution of mining power in a constructed pool of 100 miners (logarithmic scale).	53
6.8	The number of credits the top miner was reset to for every mined block over 200 000 mined blocks in a 10, 100 and 1 000 miner pool.	54

6.9	The credits a miner has been reset to for 200 000 mined blocks, sorted in descending order for pool sizes of 10, 100 and 1 000 miners.	55
6.10	The average credits a top miner is reset to after 200 000 mined blocks for pools of various sizes of miners that have been generated from sampling from a log normal distribution of hash rates.	56
6.11	The variance of the credits a top miner is reset to for a simulation of 200 000 blocks for pools of various sizes.	57
A.1	The top ten Ethpool miners by hash rate for a given time on 06-09-2017.[?] .	67
A.2	The distribution of the logs of the hash rates for 729 miners in Ethpool resembling a normal distribution.	68
A.3	The distribution of the logs of the hash rates for a constructed pool of 1 000 miners, generated via sampling.	68
B.1	The full UML diagram of the EthSim event simulator.	70

List of Tables

3.1	The priority queue with miners of different sizes over a series of blocks.	14
3.2	A miner taking advantage of the credit resetting mechanism	15
5.1	Results for a simulation of a two miner case for 100 000 blocks	31
6.1	Results for a simulation of a 1 000 miner pool for a duration of 200 000 blocks	41
6.2	Results for a simulation of a 100 miner pool for a duration of 200 000 blocks .	42
6.3	Results for a simulation of a 10 miner pool for a duration of 200 000 blocks . .	42
6.4	The results of the ExpVal1 strategy used in a 10, 100 and 1000 miner pool compared to the rewards received when no strategy is pursued in a simulation of 200 000 blocks.	45
6.5	A theoretical example of computing the expected end of round credit payout relative to the cost of giving up a number of rounds	48
6.6	The results of the ExpVal2 strategy used in a 10, 100 and 1 000 miner pool compared to the rewards received when no strategy is pursued over a period of 200 000 blocks.	49
6.7	The results of the Hopping strategy used in a 10, 100 and 1 000 miner pool compared to the rewards received when no strategy is pursued over a period of 200 000 blocks.	51

Chapter 1

Introduction

1.1 Motivation

Ever since Satoshi Nakamoto first introduced the concept of Bitcoin in 2008, the cryptocurrency landscape has turned into a \$163 billion¹ digital gold mine [17]. This can be accredited to certain features Bitcoin first introduced, perhaps the most notable, a decentralised consensus mechanism. The consensus mechanism is rooted in network nodes, called miners, trying to solve computationally intensive cryptographic puzzles where the difficulty is determined by the network. Miners are incentivised by rewards given to the miner who solves the puzzle, a compensation for his invested computational effort. By making use of a publicly accessible distributed ledger commonly known as the blockchain, Bitcoin ultimately gave rise to an entire new ecosystem of currencies which can cut out any form of central authority, yet maintain the level of trust the later provides in centralised systems.

Cryptocurrency mining remains a fundamental structural component for the effective workings of the decentralised consensus mechanism. However, rising difficulty levels of the cryptographic puzzles underpinning the mining process have posed severe constraints on the frequency of rewards paid to individual miners. Hence, cryptocurrency mining pools have become a favorable means for the reduction of the high variance individual miners face. A pool operator tracks the work each miner of the pool contributed and applies some type of reward payout scheme in order to distribute block rewards. Various cryptocurrency mining pools have evolved over the years, making use of several different reward payout schemes.

A relatively new reward payout scheme has been introduced and implemented by Ethpool, a popular Ethereum mining pool, and shall be referred to as a *queue-based* reward payout scheme. In such a scheme, the pool operator tracks the work each miner in the pool performs and rewards the contributed work by giving out credits to the miners. The number of credits a miner accumulates relative to the credit balances of the other miners in the pool determines which miner receives the next block reward and consequently has his balance reset. Even though currently Ethpool may be the only public mining pool using such a payout scheme, given the recent increases in global cryptocurrency mining activity, it is beneficial for mining pool miners of all sizes, to understand the benefits and limitations of such a scheme.

In this dissertation the workings of queue-based mining pools will be examined, as well as any

¹Total cryptocurrency market capitalisation. Source: <https://coinmarketcap.com>. Accessed: 2017-09-08

potentially exploitable aspects of the scheme. The amount of work miners of different sizes perform in a queue-based pool will be assessed and compared against their obtained reward. Additionally, the effectiveness of different mining strategies a miner could pursue in order to benefit from the non-uniform credit-reset mechanism specific to queue-based pools will be an area of focus. Furthermore, as part of this project, a simulation environment has been created and allows for the modeling of the workings of a queue-based pool. This dissertation adds to the literature on cryptocurrency mining pool reward payout schemes, by providing an analysis of the workings of queue-based mining pools and of the effectiveness of different mining strategies under such a scheme.

1.2 Aims and objectives

The aims and objectives of this dissertation are to:

- Examine the workings of a queue-based mining pool and assess the fairness of the payout scheme.
- Develop a simulation framework that allows for the simulation of a mining pool that makes use of a queue-based reward payout scheme and allows for the collection of data, specifically the credits development of the pool, reward distribution, performed work, as well as the credits development of individual miners.
- Develop a simulation framework that allows for the modeling of a scenario in which more than one queue-based mining pool exists.
- Propose and simulate, using the simulation framework, different mining strategies a miner could potentially employ in a queue-based mining pool to increase his payout.
- Identify under what conditions mining strategies can be employed effectively to increase the number of blocks rewarded to a miner.

1.3 Contributions

This dissertation examines the workings of a queue-based cryptocurrency mining pool and discusses the effectiveness of mining strategies which may be pursuable by miners with above average hash rates in order to increase their earnings.

The specific contributions of this dissertation are described below:

1.3.1 A simulation framework for queue-based cryptocurrency mining pools

A queue-based cryptocurrency mining pool simulation framework called *EthSim* has been developed in C++ using object-oriented design. The simulation environment allows for the simulation of the normal workings of a queue-based mining pool as well as for the fair comparison of different mining strategies a miner could employ under a queue-based payout scheme. Furthermore, the simulator allows for the modeling of multiple different queue-based mining pools and even for the theoretical effects of pool hopping between two or more queue-based pools.

A user of *EthSim* can configure simulations via a detailed configuration file and obtain relevant data for every simulation specified. Obtainable data includes but is not limited to the development of one or more miners' credit balances, the credit development of one or more mining pools over a certain number of blocks mined, the luck of a pool for a given number of rounds, the work a miner performed per rewarded block, as well as the work-payout ratio of a miner in terms of the number of blocks a miner was rewarded by the pool compared to the number of blocks for which he solved the cryptographic network puzzle, i.e. which he mined.

1.3.2 Potential issues with a queue-based payout scheme

By examining the workings of a queue-based pool, particularly the rewards a miner receives compared to the work he performs, it has been identified from joint work with William J. Knottenbelt, Alexei Zamyatin, Katinka Wolter, Peter G. Harrison, and Catherine E.A. Mulligan, that miners with above average hash rates are disadvantaged by the scheme as they perform on average more work per block than small miners. However, mining strategies have been proposed whereby a miner with an above average hash rate could potentially increase his earnings through share withholding, the donation of blocks, or the use of multiple payout addresses given that specific conditions prevail.

1.3.3 A comparison of mining strategies in queue-based mining pools

The simulation framework *EthSim* has been designed in a way to allow for the easy implementation of new condition-based mining strategies. The design of the simulator is very modular and thus allows for the easy implementation of additional mining strategies by specifying a new condition and linking it to an appropriate action a miner may pursue when generating a share. Condition-based strategies allow for the configuration and specification of a particular condition, i.e. a certain queue constellation, which is checked by a specific miner while he submits shares to the pool operator. Each time the condition is active, a miner pursues an action related to the share submission process, i.e. donate the share to some other miner. Multiple strategies have been proposed, simulated and evaluated in this dissertation. It has been found that a mining strategy that accounts for the other miners' hash rates and credit balances, the network and share difficulty, as well as the duration of the current pool round by computing expected end of round queue constellations, can be used to potentially increase a miner's rewards.

1.3.4 Limitations of mining strategies in queue-based mining pools

The effectiveness of the different proposed mining strategies across queue-based pools of various sizes has been found to be affected not only by the size of a pool but predominantly by the distribution of hash rates in a pool. Simulation results have shown that a high variance and average reset balance may pose favourable conditions for attacking miners aiming to exploit the credit resetting mechanism. However, it has been found that effectiveness diminishes as the distribution of hash rates for a pool approaches a log normal distribution, the assumed distribution of hash rates in a queue-based pool, as observed in Ethpool. A mining pool in which the hash rates of the miners are log normally distributed implies that end of round

credit differences between the first and second ranked miners are most likely not large enough for an attacking miner to benefit from intentionally missing out on a block reward.

1.4 Outline

The remainder of this dissertation is organised as follows:

Chapter 2 explains the concept of cryptocurrencies and the workings of cryptocurrency mining. The notions of solo and pooled mining are introduced, leading up to the examination of traditional mining pool reward payout schemes, such as Pay-Per-Last-N-Shares (PPLNS) and proportional payout schemes.

Chapter 3 examines a new queue-based reward payout scheme as first implemented by Ethpool, a popular Ethereum mining pool. Potential issues rooted in an underlying credit resetting mechanism of the scheme are discussed. A theoretical scenario whereby a large miner may increase his credit balance by intentionally not winning a round is provided and hence gives the motivation for examining these potential reward-increasing mining strategies further.

Chapter 4 outlines the design and implementation of EthSim, an event-based simulation framework that allows for the modeling of different queue-based mining pool scenarios. The simulator is presented by primarily explaining the design and implementation of the most important features and components, as well as discussing the underlying motivations for the overall system design.

Chapter 5 assesses the workings of a two miner pool and specifically evaluates the work performed and blocks received by a large miner that employs a variation of mining strategies. The simulation set up is explained and data obtained from a mining pool simulation of 100 000 blocks is analysed. The potential risk of misinterpreting the performance of these strategies in a two miner pool is pointed out and reasons as to why strategies should be examined in a multi-miner case before drawing any conclusions about their effectiveness, are given.

Chapter 6 proposes and explores the performance of different mining strategies, primarily strategies which are aimed at benefiting from the scheme's underlying credit reset mechanism. The same strategies that were pursued in the two miner pool are examined in a pool of 10, 100 and 1 000 miners. In order to account for factors specific to queue-based mining pools, mining strategies that aim to make use of expected end of round queue constellations are proposed and their performance is assessed. Additionally, a theoretical mining strategy based on the notion of pool hopping, assuming a second queue-based pool exists, is proposed and simulated. Lastly a thorough evaluation of the effectiveness of all considered mining strategies across the different pool sizes is provided.

Chapter 7 concludes this MSc dissertation by summarising and evaluating the achievements presented and highlighting potential opportunities for future work.

1.5 Statement of originality and publications

I declare that this thesis was composed by myself, and that the work that it presents is my own except where otherwise stated.

The following publication arose from work conducted during the course of this thesis:

- **IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems 2017 (MASCOTS)** [20] presents that cryptocurrency miners with above average hash rates are disadvantaged by the underlying mechanism of a queue-based reward payout scheme as first implemented by Ethpool. Furthermore, it is examined how three different mining strategies can be pursued by a large miner to increase his earnings in a hypothetical two miner scenario. This is joint work with William J. Knottenbelt, Alexei Zamyatin, Katinka Wolter, Peter G. Harrison, and Catherine E.A. Mulligan.

Chapter 2

Cryptocurrency mining, pools and reward schemes

This section explains the workings of cryptocurrencies and the underlying concept of cryptocurrency mining. Additionally, the different types of mining pools and reward schemes that have evolved over the years are presented.

2.1 Cryptocurrency mining

With more than 1 100 alternative cryptocurrencies¹, or *altcoins*, the modern landscape for cryptographic currencies has developed tremendously since the introduction of Bitcoin [17] in 2008. Bitcoin is a decentralised cryptocurrency that consists of a peer-to-peer network, where different nodes may conduct *transactions*. Transactions represent the transfer of some token, i.e. Bitcoin, from one wallet address to another and are stored in blocks on a distributed public ledger known as the *blockchain*. A *block* is a data structure containing a list of transactions between nodes in the network. Each block has a unique header, which contains, amongst a timestamp and other fields, a hash of the header of the previous block, and is hence “chained” to it. A visualisation of this relationship is presented in Figure 2.1, showing how all blocks are linked via the hash of a previous block’s header.

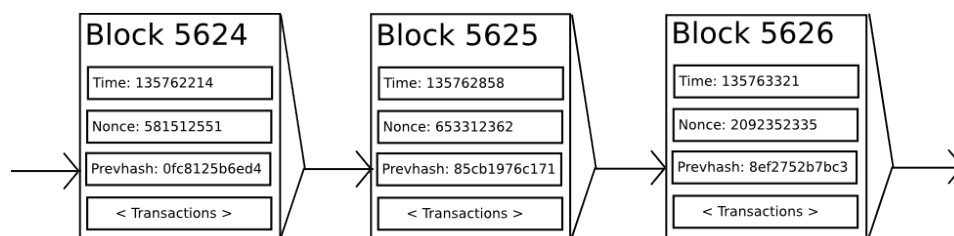


Figure 2.1: A representation of a blockchain and the relationship between individual blocks [7].

¹Source: <https://coinmarketcap.com>. Accessed: 2017-09-04

Once a block is appended to the blockchain, the transactions included get confirmed. A potential problem with a decentralised log of transactions is the restriction of network participants from double spending the same coins. As every node maintains a copy of the blockchain this can be overcome by including only non-conflicting and valid transactions in each block. However, this raises the issue of determining which node gets to append the next block of transactions to the blockchain, as well as the problem of identifying the true state of the blockchain given that there is no central authority governing this process.

In order to overcome these potential issues, Bitcoin introduced a new consensus mechanism which allows for all network participants to determine the latest state of the blockchain and hence avoids using a central authority [17]. This mechanism, also referred to as *Nakamoto consensus*, which decides on the miner who gets to propagate a block through the network and hence append the next block to the blockchain is governed by the *Proof-of-Work* (PoW). The PoW is a computationally intensive puzzle that will be solved by some node in the network in a process referred to as *mining*. *Miners* are participating nodes that receive and accumulate transactions from all other network participants and store these in a block, by solving the PoW puzzle. In order to determine which transactions should be included by a miner in a block, miners evaluate the *transaction fee* associated with every transaction, as these will be collected by the miner who gets to append his block to the chain.

Proof-of-Work algorithms are generally designed such that there is no better strategy to find a valid nonce than trial and error, as this process is cheap and trivial [12]. In Ethereum, the second largest cryptocurrency after Bitcoin in terms of market capitalisation², the PoW uses the *Ethash* algorithm, which requires a miner to find some nonce input to the algorithm, such that the result is below a certain value that is determined by the PoW difficulty, commonly referred to as the *network difficulty* [11]. When a miner computes a *hash*, a potential solution to the puzzle, he will instantly see whether his solution is valid or not. Each tried nonce, or computed hash, is a Bernoulli trial where the success probability is determined by the set difficulty [13]. Therefore, the number of attempts it takes a miner to find a valid solution to the PoW puzzle and consequently *mine* a block, is random with a geometric distribution.

Finding a valid block is a Poisson process, where a miner's probability of finding a valid solution to the PoW puzzle in some period of time is determined by rate parameter $\frac{h}{D}$ where h is the miner's computational power, or *hash rate*, and D the network difficulty. The hash rate of a miner is the number of nonces tried per second and is typically measured as megahashes (MH) per second, where $1 \text{ MH/s} = 10^5$ hashes per second. In Ethereum, the difficulty dynamically adjusts itself such that the network finds a block every 12 seconds on average [12]. However, it is important to note that the mining process is memoryless, as there is no guarantee that a miner will find a block at the time he is expected to, which explains the high variance individual miners face.

The role of the PoW and hence mining activity conducted by the miner nodes is to establish a decentralised emergent consensus. Miners who invest their computational resources to try to solve the PoW implicitly allow for all nodes in the network to independently select the blockchain with the largest demonstrated computational effort [6]. Therefore, each time a miner node finds a valid solution to the PoW problem, he appends the next block containing transactions to the blockchain and will be rewarded for his computational effort a fixed amount of newly minted units of the currency. This is referred to as the winning node having *mined* units of the currency and consequently increased the overall volume of it. As all

²Source: <https://coinmarketcap.com>. Accessed: 2017-09-02

nodes in the network will receive this new block, everyone can agree on the same state of the blockchain.

The emergence of Ethereum had a big impact on the cryptocurrency mining community. Ethereum introduced a Turing-complete scripting language that allows for the creation of so called *smart contracts*, programs that govern the transfer of *Ether* (ETH), which is the underlying currency in Ethereum. This essentially allows for an easy creation of on-blockchain token systems, and hence played a significant role in the rise of many altcoins. Apart from the increase in the number of altcoins, the cryptocurrency landscape has experienced tremendous increases in overall market capitalisation since the first introduction of Bitcoin and Ethereum. Figure 2.2 shows the top ten cryptocurrencies by market capitalisation, which may justify why cryptocurrencies are often referred to as digital gold.




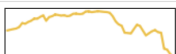

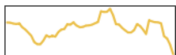







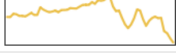






▲#	Name	Market Cap	Price	Circulating Supply	Volume (24h)	% Change (24h)	Price Graph (7d)
1	 Bitcoin	\$68,685,102,045	\$4151.70	16,543,850 BTC	\$2,643,250,000	-8.02%	
2	 Ethereum	\$26,721,356,238	\$282.98	94,427,426 ETH	\$1,490,140,000	-17.06%	
3	 Bitcoin Cash	\$8,444,142,780	\$509.91	16,560,163 BCH	\$339,253,000	-15.59%	
4	 Ripple	\$7,578,967,099	\$0.197658	38,343,841,883 XRP *	\$206,779,000	-11.99%	
5	 Litecoin	\$3,269,726,571	\$61.95	52,778,982 LTC	\$751,661,000	-19.12%	
6	 NEM	\$2,349,423,000	\$0.261047	8,999,999,999 XEM *	\$8,183,090	-12.27%	
7	 Dash	\$2,313,419,743	\$306.92	7,537,582 DASH	\$37,567,700	-12.12%	
8	 IOTA	\$1,624,440,883	\$0.584430	2,779,530,283 MIOTA *	\$62,254,000	-19.42%	
9	 Monero	\$1,554,862,883	\$103.42	15,035,178 XMR	\$84,656,900	-14.28%	
10	 Ethereum Classic	\$1,478,870,148	\$15.54	95,194,793 ETC	\$227,460,000	-16.77%	

Figure 2.2: Screenshot: the top ten cryptocurrencies by market capitalisation

3

2.2 Solo mining

Individuals mining independently are generally referred to as *solo miners*. As previously mentioned, finding a block as a solo miner with some constant hash rate h follows a Poisson process where $\frac{h}{D}$ is the rate parameter. When a solo miner mines for time t he receives on average $\frac{ht}{D}$ blocks. The amount of blocks found follows a Poisson distribution where the expected number of blocks is $\lambda = \frac{ht}{D}$. The expected payout of a solo miner would thus be $\frac{htB}{D}$, where each found block has an associated block reward B [18]. The miner's expected revenue per hashing operation can be formulated, as stated by Rosenfeld [18], as

$$E[R_h] = \frac{R_e}{D}$$

and the miner’s corresponding variance as

$$\text{Var}[R_h] = \frac{R_e^2}{D}.$$

2.3 Mining pools

A *mining pool* refers to a group of miners that pool together their computational resources in order to increase their overall chance of finding a block. This allows individual miners to earn a steadier payout stream of rewards as they reduce the variance between finding blocks. The pools total hash rate $H = \sum_{i=1}^n h_i$ is the sum of the hash rates of all n individual miners that make up the pool. Therefore the pool expects to find $\frac{Ht}{D}$ valid blocks over time period t [18].

Each mining pool has a pool operator, who charges a proportional fee f each time a block reward B is received and splits up the remaining reward $(1 - f)B$ based on the reward payout scheme implemented by the pool. In order for the pool operator to be able to assess every miner’s contributed computational work, each pool member receives a computational puzzle with a lower difficulty than the network difficulty. Every valid solution to the pool’s puzzle is referred to as a submitted *share* by a miner. Therefore the submitted number of shares by a miner is on average proportional to the number of hashes he computed while attempting to find a block. The process of computing a share is identical to the process of trying to find a valid block apart from the lower difficulty. The pool operator consequently tests whether every submitted share is also a valid share to the network’s problem. Should a submitted share also be a valid share (a solution to the network problem) then a block has been mined by the pool.

2.4 Miner rewards and reward payout schemes

In Ethereum a block reward consists of 5 ETH, in addition to the total fees stemming from the transactions in the block [8]. However, in Ethereum, one has to distinguish between full blocks and *uncle* blocks. An uncle block is a block found, which does not become the new head of the blockchain. This may occur when a competing miner finds a block and benefits from his faster network connectivity, allowing his block to be propagated at a faster rate through the entire network and consequently become the new head of the chain. However, unlike Bitcoin, Ethereum rewards these uncle blocks and currently issues 3.75 ETH per uncle. The probability of finding an uncle can be denoted as p_u and hence the expected reward per mined block can be denoted as

$$R_e = R_b(1 - p_u) + R_u p_u.$$

It should also be noted that potential risks for mining pools have been identified. Rosenfeld [18] introduces the notion of *pool-hopping* attacks, whereby miners may submit shares to different pools based on favorable conditions in order to try to increase their rewards. An examination of potential problems rooted in the prevention of such pool-hopping attacks was conducted by Lewenberg et al. [15]. Further research on possible attack scenarios between different mining pools has highlighted potential risks, such as possible block withholding [13], [9], [16], and denial-of-service [14] attacks. Furthermore, Schrijvers et al. have shown that under proportional reward schemes, miners may deliberately hold on to a PoW solution for a

temporary period of time in order to increase their payout, consequently harming the pool in which they mine[19]. In this dissertation, the employed term *attacks* refers to the potential exploitation of a queue-based mining pool’s reward payout scheme. A mining pool’s *reward payout scheme* refers to the mechanism which is employed by a pool operator to distribute mined block rewards amongst the members of the pool.

Deciding on how to split up and allocate the block reward in a fair manner amongst the members of a mining pool is not a trivial problem and may very well differ between pools. Over the years different methods for splitting up rewards have been implemented by mining pools. The most conventional of reward distribution schemes are proportional payout, pay-per-share (PPS) and pay-per-last-n-shares (PPLNS). The following subsections will provide a more detailed analysis of each reward payout scheme, based on Rosenfeld’s [18] analysis of reward systems.

Proportional payout scheme

The proportional payout scheme is perhaps the most simple and intuitive approach towards splitting up the block reward. This scheme is rooted in the concept of *rounds*, whereby a round constitutes to the time between one block found to the next one [18]. As mentioned in the previous section, miners in a pool compute and submit shares in order to prove that they are performing work. The proportional scheme suggests that a miner’s claim of the block reward B should be proportional to his number of shares submitted n as a fraction of the total number of shares collected N by the pool in a given round [18]. Thus, a miner’s payout per round should be equal to

$$\frac{n}{N}(1 - f)B.$$

A problem rooted in the proportional payout scheme is that this scheme only works well under the assumption that the number of miners in the pool is fixed [18]. In reality, however, a pool using a proportional system might be subject to pool-hopping attacks. Pool-hopping may occur when a miner leaves a particular mining pool if the pool has been unlucky for a longer period of time and thus the expected reward of the miner is lower due to the increased number of shares that have been submitted [9]. This may be reason enough for the miner to join a different pool in a more favorable state. Rosenfeld [18] finds that when the shares submitted are 43.5% of the network difficulty the point has been reached where the fair average payout is equal to the expected payout. Furthermore, Rosenfeld suggests that a pool-hopper will only mine before that point and subsequently leave the pool, before returning at the start of a new round. This may seem more intuitive when one thinks about the possibility of finding blocks early on in a round and that short rounds with fewer shares lead to higher revenue per share, whereas long rounds decrease share utility.

Pay-per-share

PPS is one of the most common implementations of proportional payout scheme, in which the mining pool operator absorbs all the variance individual miners are exposed to by paying pool members directly for each submitted share. Thereby some miner i can expect a reward per block of

$$E[R_i] = (1 - f)R_e \frac{s_i}{\sum_{j=1}^n h_j}$$

in a pool of n miners [18]. At the start of every new round, after a block has been found, every miner's credits are reset to zero. The number of expected shares per block can be expressed as

$$E[S_b] = \frac{D}{d}$$

Although miners may still experience some variance in the number of shares found in a given time period, they do no longer face variance in the reward per share [18]. More specifically, Rosenfeld [18] further states that the variance faced by a miner of the pool is lower than had he solo mined by a factor of $\frac{D}{\ln D}$.

A major concern with this particular reward scheme is that the pool operator absorbs all the variance. Individual miners are no longer negatively affected by unlucky mining streaks experienced by the pool. In fact, the pool operator would be the only agent negatively affected and could eventually run out of funds and thereby create a situation where miners may not receive their expected payouts or even lead to the closing of the pool. To compensate such a high risk, the pool operators in PPS schemes tend to charge higher fees at the cost of the members of the pool.

Pay-per-last-n-shares

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 even if a block was not found in that period of time. This automatically takes away the incentive to only submit shares during the early period of a round. The reward is only paid out to miners of the pool when the number of shares submitted is greater than the number of expected shares, $N > E[S]$. Thereby the expected revenue for some miner i is

$$E[R_i] = (1 - f)BR_e \frac{s_i}{N}$$

where s_i is the number of shares submitted by miner i and B the number of blocks found during the last N shares [18].

Chapter 3

Queue-based mining pools: Ethpool

A new reward system has been implemented and introduced by Ethpool, which follows a slightly different approach in keeping track of how much work every miner has performed than more conventional mining pools. Each time a miner submits a share to the pool operator, he receives a number of so-called *credits* equivalent to the share difficulty. The pool operator maintains a maximum priority queue, ranking each miner according to their accumulated credit balances. An example of Ethpool's implementation of such a queue-based payout system is shown by Figure 3.1 [4]. The wallet addresses of the fifteen miners with the highest credit balances are displayed, as well as their respective hash rates, rank and estimated time until each miner receives a block.

Rank	Miner	Hashrate	Estimated TtnB	Credits
1	ae5ba84c7a77d337bc0223cdf2ca7e836881435	10.0 GH/s	Waiting for next block	2365.906t
2	bc48393a0571b77f543d3f758d8cd5ce136a974e	10.0 GH/s	in an hour	2334.299t
3	3222ffb77262b2f976ae19d695e1badafe6f4d64	4.1 GH/s	in 3 hours	2329.217t
4	14C811708d11C4d8B3Cf66be7CF23fba9E1e9ffe	2.2 GH/s	in 6 hours	2321.705t
5	f425cd9d7df5cf33575f6a364729b5eaf9e87b17	4.4 GH/s	in 3 hours	2320.027t
6	0feccc4d93fbc2fc3062d1f2819f7ff083b35b02	1.1 GH/s	in 15 hours	2310.036t
7	6a830c5afb4a4e6903d962b11b3f448238d96014	22.2 GH/s	in 42 minutes	2309.969t
8	2cd4fa97ee85dd8848a49a53b9b9a5758d7a7acd	33.6 GH/s	in 28 minutes	2309.803t
9	5cfea98867469500b90f0aa18402b95531869662	515.1 MH/s	in a day	2308.011t
10	19395847162029a1809cb298ce19d6cdd8acd59d	10.4 GH/s	in 2 hours	2307.539t
11	312907bF5Cc65Df63c81b714c6F198937535BBF9	185.9 MH/s	in 4 days	2306.099t
12	d309e8c21F56b079181f27E54fd816524c89a0F9	1.6 GH/s	in 11 hours	2302.796t
13	633a42feccefa5bc41a58b76202816a1ee8f0c	193.3 MH/s	in 4 days	2302.003t
14	f90395999E115cd9F2092dE75684297F800EB28C	2.1 GH/s	in 9 hours	2295.494t
15	1d9438829369a14553eC30f61d87A2d3220F7309	2.2 GH/s	in 9 hours	2292.941t

Figure 3.1: Screenshot: the maximum priority queue as implemented by Ethpool showing the fifteen miners with the highest accumulated credit balances.

When a block is found by the pool, the pool operator allocates the full block reward¹ to the miner residing at the top of the queue. The credit balance of the first ranked miner is

¹less pool fees

subsequently reset to the difference between his own and the second ranked miner's credits. This can be expressed formally by stating that for a first ranked miner $m_{(1)}$ and a second ranked miner $m_{(2)}$, when a block is found miner $m_{(1)}$ receives the complete block reward and his new credit balance is equal to

$$c(m_{(1)}) := c(m_{(1)}) - c(m_{(2)})$$

It should be noted that this credit resetting mechanism is non-uniform in the sense that the credits of top miners may be reset to differing start balances.

Ethpool states that in order to win a block a pool member should expect to collect D credits [3]. In case an uncle block is found rather than a full block the miner head of the queue will receive the uncle reward R_u . However, the credits of the first ranked miner will not be reset but rather the miner continues to reside at the top. Taking this into account the expected number of credits of a miner on top of the queue at a time a block is found is equal to

$$E[c(m_{(1)})] = (1 + p_u)D$$

3.1 Manipulating the queue: strategic mining

As the end of round credit differences between the first and second ranked miners vary over time, the credit balance of some miners is reset to a higher amount than for other miners. The reason as to why credit balances are not reset to zero is primarily to incentivise miners who are top of the queue to continue to maximise their work rather than to stop their mining efforts [4]. By resetting their balance to the difference between their own and the second ranked miner's balance, the *reset balance*, the top miner will not lose any *extra* work he performs.

A simple example that highlights the non-uniform credit resetting mechanism in more detail is provided in Table 3.1. The example, which is similar to one proposed by Zamyatin et al. [20], shows how two large miners, Bob and Alice, who each earn 10 credits per round, and a smaller miner, Dave, who earns 5 credits per round, are affected by the credit-resetting mechanism when a block has been found. Table 3.2a shows the initial priority queue order, where Bob is positioned on top and is expected to receive the block reward when the next block is found by the pool. Once a block has been found Bob's credit balance is reset to the difference between his and Alice's credits, namely 5 credits (Table 3.1b). The order of the miners in the queue has now changed and Alice is the new head of the queue with a new balance of 145 credits. Bob and Dave have also not stopped performing work and subsequently increased their credit balances by 10 and 5 credits, respectively (Table 3.1c). However, when the next block is won, Alice's credit balance will be reset to 70, which is a significantly higher amount than what Bob's credits were reset to after the previous round although his winning credit balance was almost identical to Alice's. Furthermore, it can be seen how Alice is now in a very favourable position to reach the top of the queue again relatively soon, even though she did not perform any work yet.

The non-uniform credit resetting mechanism could give rise to potential attack vectors for large miners, or more precisely, could allow large miners to increase their expected payout at the cost of other miners. A miner could evaluate the non-uniform expected end of round credit differences (reset balance) for the current and upcoming rounds. It could be the case that a large miner, who is expected to win the current round anticipates to receive a higher

Table 3.1: The priority queue with miners of different sizes over a series of blocks.

(a) Before Block i			(b) After Block i		
Position	Miner	Credits	Position	Miner	Credits
1	Bob	140	1	Alice	130
2	Alice	130	2	Dave	70
3	Dave	70	3	Bob	5

(c) Before Block i+1			(d) After Block i+1		
Position	Miner	Credits	Position	Miner	Credits
1	Alice	140	1	Dave	75
2	Dave	75	2	Alice	65
3	Bob	15	3	Bob	15

amount of start credits and thus intentionally decides not to win the current round. The miner that was expected to have the highest number of credits by the end of the round could employ certain actions to allow some other miner to overtake him in terms of total credits. Examining different conditions, which determine a favourable queue constellation such that this behaviour could be rewarding, will be the focus of the upcoming chapters. For now, a favourable queue constellation should be defined as: an expected end of round constellation of a maximum priority queue whereby the credit difference between the first and second ranked miners for round r_i would exceed the expected end of round credit difference between the first and second ranked miners for round r_j , where $i > j$, and the attacking miner is expected to win both rounds.

The following sections examine four different *actions* a miner could theoretically pursue at any point in time in order to allow some other miner to overtake him in the queue. It is assumed that only one miner takes such actions and that all other miners in the pool resume their mining activity at all times. These four actions are share withholding, tactical donation of mining power, the use of a second payout address, as well as pool hopping. The first three actions have been first proposed and examined together with William J. Knottenbelt, Alexei Zamyatin, Katinka Wolter, Peter G. Harrison, and Catherine E.A. Mulligan, and are also presented in the conference paper “Beneath the Surface: Swimming in Ethereum Mining Pools”, which was accepted over the course of this dissertation [20].

3.2 Share withholding

Share withholding is a strategy employed by a large miner, whereby the miner stops submitting shares in order to alter the queue constellation in his own interest, trying to achieve a favourable queue constellation. This can be seen in Table 3.2, which shows an initial constellation of a priority queue, similar to the previous example. However, for this example we make Bob out to be an *attacker*, a miner who is aware of the non-uniform credit resetting mechanism and thus determines his mining behaviour in accordance with the current queue constellation. Initially, Bob is first in the queue and hence the miner of the pool that should expect to receive the next block rewarded. Bob can expect his credits to be reset to 5, as this would be the current difference to Alice’s credits. However, when analysing the order

3.3. TACTICAL DONATION OF MINING POWER

of miners more closely, it can be seen that the credit difference between Alice and Dave is substantially greater than the amount of credits Bob would currently be reset to. Therefore, Bob may stop submitting shares, or withhold shares, until Alice has overtaken him in terms of credits (Table 3.2b).

Table 3.2: A miner taking advantage of the credit resetting mechanism

(a) Before Block i			(b) Block i is found		
Position	Miner	Credits	Position	Miner	Credits
1	Bob	140	1	Alice	145
2	Alice	135	2	Bob	140
3	Dave	70	3	Dave	75

(c) After block i			(d) Block i+1 is found		
Position	Miner	Credits	Position	Miner	Credits
1	Bob	140	1	Bob	150
2	Dave	75	2	Alice	80
3	Alice	5	3	Bob	15

(e) After Block i+2			(f) After Block i+3		
Position	Miner	Credits	Position	Miner	Credits
1	Dave	80	1	Bob	80
2	Bob	70	2	Alice	25
3	Alice	15	3	Dave	20

Once Alice has surpassed him, Bob can resume his mining activity by submitting shares again. Now, when the next block is found by the pool, Alice’s credits are reset to 5, while Bob finds himself at the top of the queue (Table 3.2c). When the next block is found and Bob’s credits will be reset to 70 he finds himself in a much more favourable position than had he not altered the initial order of the queue (Table 3.2e). In fact, if one considers the order of the queue after the next block, Bob will be again at the top of the queue and thus entitled to the next reward. The reason for using a credit resetting mechanism as implemented by Ethpool, rather than resetting the credits of the top miner to zero when a block is found, is to encourage the top miner to continue to submit shares given that his extra work will not be lost. However, this simple example demonstrated that by making decisions based on the expected credit difference, one miner, Alice, has been cheated out of a significant number of credits, which consequently affects her pool earnings in a negative way.

3.3 Tactical donation of mining power

A drawback of share withholding is that the large miner employing that strategy, does not only fail to maximise his share submission, but furthermore does not perform any work at all while he waits to be overtaken by a different miner. When waiting to be overtaken by a different miner, rather than doing no work and missing out on potential credits, a large miner could donate his mining power to the potential overtaker. This ideally speeds up the overtaking process, which may be beneficial to the attacker if the pool is in a situation, where

it is expected to find a block relatively soon. Such a scenario has in fact been observed in Ethpool by Zamyatin et al. [20] and is shown in Figure 3.2.

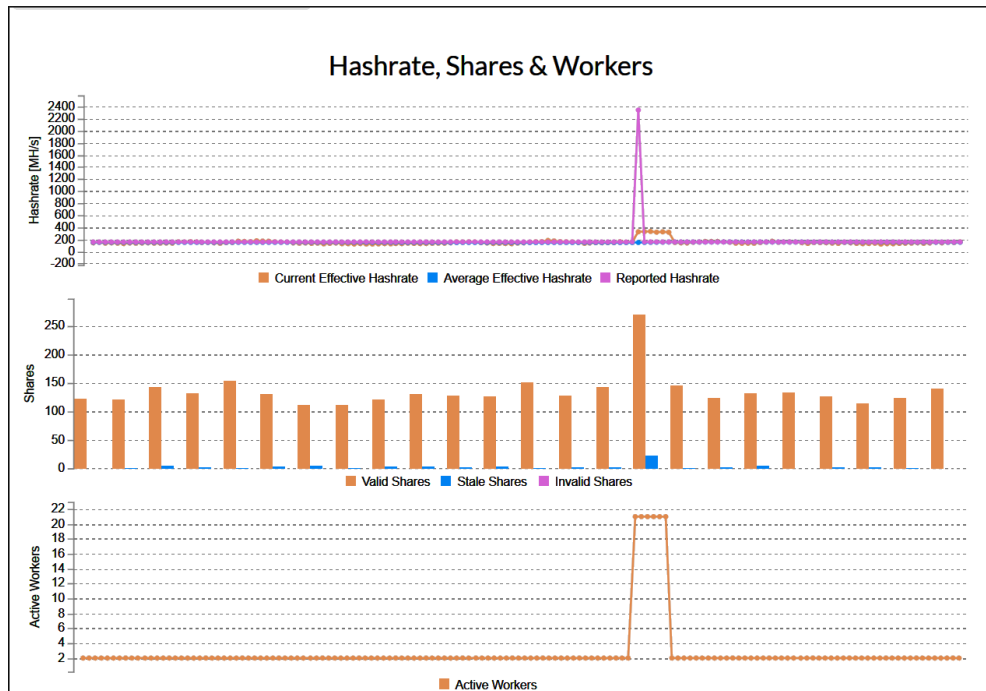


Figure 3.2: Screenshot: A miner in Ethpool experiences a sudden increase in his hash rate for a short period of time [20].

It can be seen how a miner whose average hash rate remains at 200MH/s for long periods of time, suddenly receives a significant boost in the number of his active workers, his associated mining hardware, and hence an increase in his overall hash rate. The miner’s hash rate increases from 200MH/s to almost 2.4GH/s for a short period of time before it falls back to 200MH/s. This may likely be a real world example of how a miner in a queue-based mining pool has received an unexpected donation of computing power, potentially benefiting some other miner.

The reason for why the donation of shares is even possible in the first place is due to how miners inform pool operators about which payout address, or *wallet*, should receive credits for their performed work. Any miner could use any payout address and inform the pool operator to reward the credits to the respective address. This suggests that the pool operator is unable to differentiate between mining power having been tactically allocated or actually belonging to a specific miner. Therefore the attacker cannot be prohibited from donating his power in the form of shares until the condition that he envisions has been met.

3.4 Second payout address (wallet)

The third action, which may be pursued by a larger miner, is similar to the proposed tactical donation of mining power. However, rather than giving away credits, or performed work, to the overtaker, the attacking miner could set up a second payout address (wallet). Hence when an attacker waits to be overtaken by some miner, rather than donating his shares to a different miner, and thereby giving away his performed work, he could submit shares to

his second wallet. This method allows an attacking miner to alter the queue constellation without having to stop lose, or give away, any of his work efforts to some other miner.

3.5 Pool hopping

A fourth action that is based on the notion of pool-hopping should be examined. Pool-hopping, as discussed by Rosenfeld [18], is the exploitation of different levels of attractiveness of mining across more than one pool. If a mining pool appears unattractive in terms of variance and expected earnings at a particular point in time, a miner may be able to increase his expected rewards should he switch to a pool whose attractiveness for mining is rather high. However, this is damaging to all members of the pool who mine on a continuous basis. Rosenfeld [18] therefore states that only pools resistant to pool-hopping or a system where all miners mine solo are sustainable.

Even though Ethpool is the only queue-based mining pool at the time of writing, throughout this dissertation the hypothetical scenario in which multiple queue-based mining pools exist should also be accounted for. An action a miner could thus take is leave the current queue-based pool and join a different one, or stop the submission of shares in one pool and redirect work efforts to a different pool. Thus, each time a specified condition is met in pool p_1 , a miner would submit shares to his other wallet in pool p_2 .

Chapter 4

EthSim: a mining pool simulator

This section outlines the design and implementation of *EthSim*¹, the simulation environment, which has been built as part of this dissertation in order to examine the workings of queue-based mining pools and the effectiveness of different mining strategies. The initial aims for the simulator are stated prior to reviewing the actual design and implementation of the system. The later will be conducted by addressing each of the specified aims. Simulation results of queue-based pools and mining strategies are examined throughout the later chapters.

4.1 The aims for the simulator

This dissertation focuses on examining the workings of queue-based mining pools, as well as the potential exploitability of the underlying credit reset mechanism via strategic mining. Hence the aim for designing a queue-based mining pool simulator was split into two main parts:

- The simulator should allow for the modeling of the workings of queue-based mining pools assuming all miners in the pool maximise their computational work efforts. A user of the simulator should thus be able to obtain data of a pool's performance and of any miner in a pool. This data should include the performed work per block by a miner, the credit development of a miner, as well as the payout ratio of blocks rewarded to blocks mined for both.
- The simulator should allow for the evaluation of the effectiveness of different mining strategies a miner could employ. More specifically, simulating the actions discussed in the previous chapter in queue-based mining pools of *any* size should be accounted for. Furthermore, the simulator should ensure that different mining strategies can be compared in a fair and unbiased manner.

¹The name *EthSim* was chosen given that Ethpool introduced the queue-based reward scheme and that the simulator should be able to effectively model the workings of Ethpool

4.2 Design

The following subsections will address the design choices made for each of the specified aims.

4.2.1 Simulating a queue-based mining pool

In order to simulate the cryptocurrency mining process in a pool, the choice for designing an event based system has been made, where events represent valid shares submitted by miners of a pool. An instance of a share event has a timestamp representing the time for when the share is found and can either be a network share event or a mining pool share event. Additionally, every share event has a reference to the miner who it was scheduled by. A network share event represents a miner having submitted a valid share to the pool's problem, which is also a valid solution to the overall network problem and thus the pool mines a block. A mining pool share event represents a miner having submitted a share that is a valid solution to the mining pool's problem but not to the network problem. The rate at which a miner schedules shares is dependent on the hash rate of the miner.

An event queue contains all scheduled events ranked by their timestamp in ascending order. The event with the lowest timestamp will be executed and the execution of the event depends on the type of share, i.e. whether it is a network or mining pool share event. Before the execution of the next share in the queue, the miner who scheduled the current executed share will schedule a new share. His new share will be inserted into the appropriate slot in the event queue, based on the share's timestamp. The outlined share scheduling and execution mechanism is visualised by a simple flow chart in Figure 4.1.

The time intervals between shares submitted by miners in the pool are created as a randomly generated number following an exponential distribution with rate parameter $\lambda = \frac{h}{d}$, where d is the share difficulty of the pool and h a miner's hash rate. For every submitted share by a miner, he receives an increase in his credits equivalent to the share difficulty. When a network share is being executed, the miner that submitted the share receives the credits for the submitted share. Additionally, the miner with the highest accumulated number of credits in the pool is rewarded a block and has his credit balance reset to the difference between his and the second ranked miner's credits. At all times there can be no more than one event share per miner in the event queue.

After every executed share the time of the simulation, which is used by miners to schedule their shares, is set to the timestamp of the most recent executed share event. The number of blocks that have been found, or mined, by a pool, are also recorded. Once the simulation has run for the specified number of blocks found by the pool, no more shares are being executed or scheduled.

4.2.2 Simulating mining strategies

For a fair comparison between different mining strategies, one has to compare the effectiveness of each strategy employed over the same sequence of events. For example, had a miner pursued a share donation strategy, in order to evaluate its performance, one has to compare it to the scenario in which a miner did not employ the strategy. Hence, the same sequence of

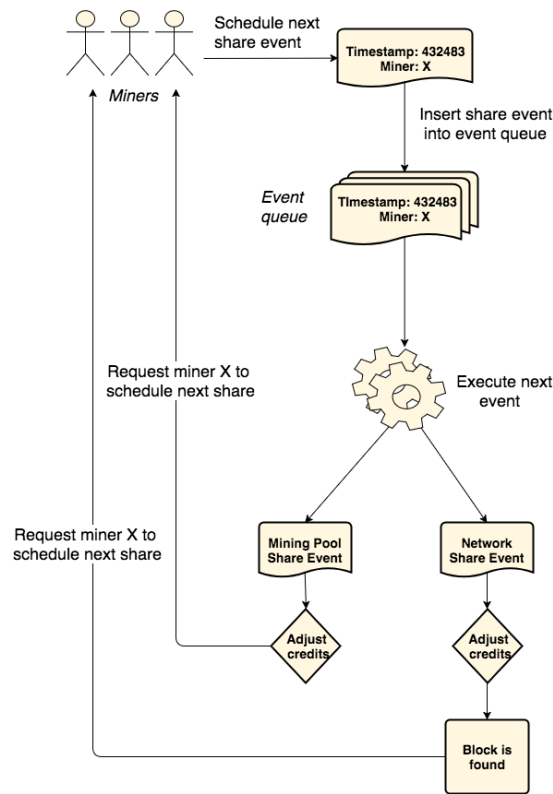


Figure 4.1: A visualisation of the share scheduling and execution mechanism.

events should be used for the simulation of different mining scenarios, as it would be unfair to compare mining strategies that have been employed over event sequences which considerably differ. A simulation of a mining pool with N miners, where $N > 1$, and no miner pursues a strategy should be referred to as the *normal* scenario. In order to evaluate the effectiveness of different mining strategies on a fair basis, a new scenario using the same sequence of events as the normal one, should be simulated for every proposed strategy.

In cryptocurrency mining, regardless of the mining pool, an individual miner can only affect his own actions, i.e. the submission or withholding of shares. Whether an attacker miner withholds, or donates his shares to some other miner or to his second wallet does not have any direct effect on the share generation of the other miners in the pool. Under the assumption that all non-attacking miners maximise their mining efforts at all times, different actions employed by an attacker would never have any direct effect on the share scheduling of other miners. Even for the attacker, the assumption is that the timestamps for his scheduled shares should be the same for all scenarios. Thereby the only difference between different mining scenarios would potentially be the handling of the attacker's shares, i.e. whether a share should be submitted and, if so, to which payout address.

Comparing different mining strategies therefore involves the simulation of multiple identical pools over the same sequence of events. The share scheduling and execution mechanism, as proposed in the previous subsection, would remain unchanged. Allowing for the simulation of different mining scenarios proved to be the most challenging aspect of designing the simulator system. As the simulator is event-based a choice had to be made of whether the share structure should be altered, such that every mining strategy scenario has its own set of shares and could thus be simulated independently. However, such a design would go against

what has been discussed so far, namely that the event sequence should be the same for all simulated mining scenarios. Therefore, rather than altering share structures, every miner in the pool keeps track of his performance across all scenarios. Each time a share event is executed, the miner who scheduled the event adjusts his credits accordingly for every scenario. Should the miner be an attacker pursuing a mining strategy then he may employ some action with regards to the handling of the share for the respective scenario.

A miner thereby keeps count of his credit balances for every scenario, as well as other important information, such as the number of shares submitted, number of blocks received and number of blocks mined. The number of *blocks mined* refers to the number of network shares, or solutions to the network problem, a miner submitted/found. Therefore, the number of blocks found also represents the number of blocks a miner would have received had he mined solo. The number of *blocks rewarded* refers to the number of blocks a miner was actually rewarded by the pool operator, i.e. the number of times the miner was top of the queue when a block was mined by the pool.

For every attack strategy to be examined, a new scenario should be added to the simulation. The only difference amongst all scenarios for a simulation is the share handling of the attacking miner, which consequently may affect his own, as well as other miner's performance. A *condition and action scheme* has been proposed to allow for the construction of mining strategies. Under such a scheme, the miner pursues a particular action, or behaviour, given that a particular condition is met. Should the condition be false then a default action should be undertaken. For example, a *condition* may refer to a particular queue-constellation or a specified credit balance the succeeding miner should not exceed. A theoretical example would be: *if miner Eve is top of the queue and the second ranked miner's credit balance is greater than or equal to 90% of Eve's total credits then the condition is true*. An *action* refers to the behaviour undertaken by the attacking miner should the condition be true and may be any of the previously proposed attack actions. Thus the term *mining strategy* in a queue-based mining pool setting can be defined as a specified action or set of actions that are undertaken by an individual miner provided that some condition is met.

A condition should be checked when the share to be executed by the event queue is a share that has been submitted by an attacker. For every scenario, apart from the normal scenario, there can be one condition per scenario, apart from the normal scenario. Thereby, each time an attacker share is executed, for every scenario the associated condition will be checked and subsequently determine whether the share should be submitted normally and increase the attacker's credit balance for the respective scenario, be withheld, or donated to some other payout address. The default action for an inactive condition is the same as under the normal scenario, namely the submission of the share and the associated increase in the credits of the attacker for that scenario. Non-attack miners should not be concerned with the condition and action scheme but rather receive the appropriate amount of credits per share for every scenario they are part of. By following this methodology, the main aspect that may differ over time between different scenarios is the queue constellation, as well as the credits and rewards miners have been reset to and received, respectively.

4.2.3 System design

An overview of the design of the simulator system is provided by Figure 4.2. This UML of the simulator only shows the relationships between all the different classes of the system in

order to highlight the system’s complexity. A full UML of the system showing class member variables and methods can be seen in Appendix B.1.

The two main components of the system are the generation and execution process of event shares and the previously explained condition and action scheme. The following sections examine the design and implementation of these two components in further depth and highlight the interaction between them. Lastly, the simulation configuration, data collection and pool populating mechanisms are explained.

4.3 Implementation: a queue-based mining pool

The implementation of the share event scheduling and execution mechanism can be examined by focusing on two key components of the system: the event queue and events, as well as the scheduling of shares.

4.3.1 Event queue and events

The simulation has at most one event queue, which uses a minimum priority heap to find the event with the lowest timestamp. The event queue maintains the time of the most recent executed event, which is taken into account when future events are being scheduled. The implemented share events are mining pool share events and network share events, both being implemented as classes of the same name, derived from a `ShareEvent` class. How an event is executed depends on the type of event and thus the `Event` method `execute()` is declared as a pure virtual method. In an instance of a network share event, a share is a valid solution to the network problem and thus not only are the credits of the miner who submitted the share increased, but in addition a block is rewarded to the pool operator, who passes it on to the miner with the most accumulated credits. Executing a mining pool share event simply results in an expected increase in the credits of the miner who scheduled the share. A UML diagram highlighting the relationship between the event and event queue classes is presented in Figure 4.3.

4.3.2 Share scheduling

Share scheduling is one of the most essential components of EthSim. As previously mentioned, a miner schedules a new share event as soon as he has no scheduled event left in the queue. As previously explained, finding a solution to the mining pool problem is also a Poisson process with rate parameter $\lambda = \frac{h}{d}$, where h is an individual miner’s hash rate and d is the share difficulty of the mining pool. The probability for a pool share to also be a valid solution to the network problem is given by $\frac{d}{D}$, where D is the network difficulty. Therefore the type of a share is determined by randomly generating a floating point value between 0 and 1 and checking whether it is below the probability of the share being a network share. Should this be the case then the share to be scheduled will be a network share event, else it will be scheduled as a mining pool share event. For the pseudo-random number generation the `drand48()` function is used, as it returns non-negative, floating-point values that are uniformly distributed over the interval [0.0, 1.0) [1].

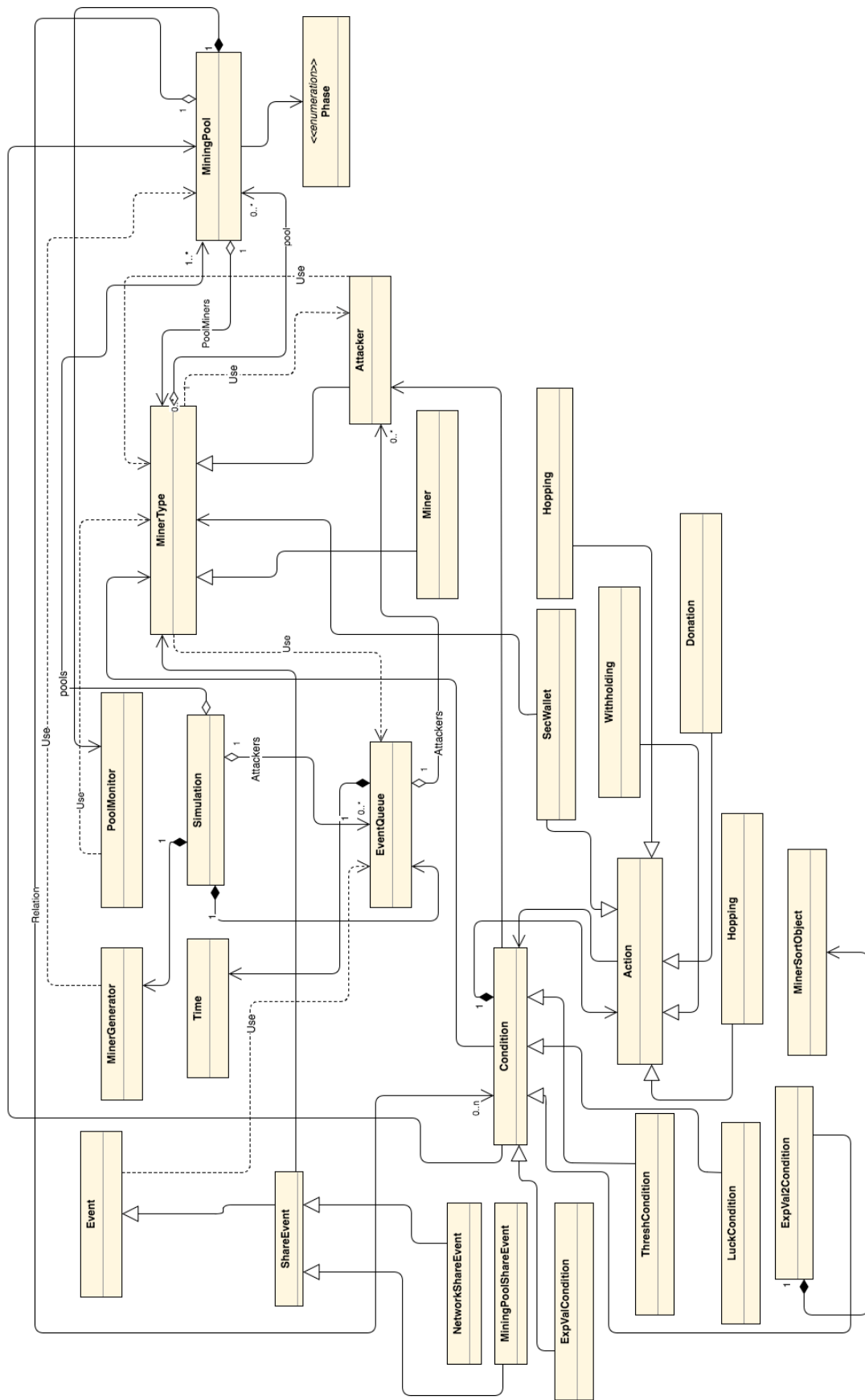


Figure 4.2: A UML diagram showing the relationships between the different classes of EthSim.

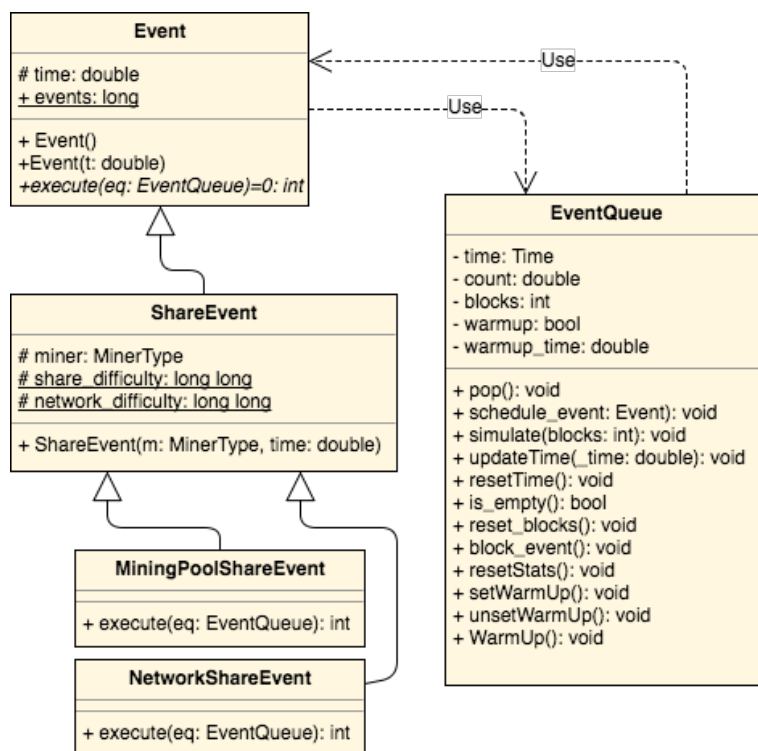


Figure 4.3: A UML diagram showing the relationship of the different event classes and event queue

The time intervals by which shares are submitted by miners are constructed as a randomly generated number following an exponential distribution, where the rate parameter is equal to $\lambda = \frac{h}{d}$, where h is the hash rate of the miner who submits a share and d the share difficulty set by a pool. This has been implemented as

$$t = \frac{-\log(\text{drand48}())}{\lambda} \quad (4.1)$$

In order for a miner to schedule his next share he creates the appropriate `ShareEvent` object for the time that is equal to the current time, which can be obtained from the event queue, plus the computed time for the submission of the next share, t .

4.4 Implementation: mining strategies (scenarios)

The implementation of mining strategies is presented by focusing on the interaction between the condition and action classes.

4.4.1 Conditions and actions

The choice for a `Condition` class falls back on the aim to simulate different mining strategies. As the simulator is event-based, a specified condition can be checked each time an attacking miner's scheduled share event is being executed. The way a condition is checked depends on

the condition itself. As the nature of a condition may vary, the `Condition` class has been implemented as an abstract class, where the method `check()`, which returns the active or inactive state of the condition, is implemented as a pure virtual method. For the simulations discussed in this thesis, four different conditions have been defined, these being a threshold condition (`ThreshCondition`), a luck condition (`LuckCondition`), as well as two conditions based on expected values (`ExpValCondition` and `ExpValCondition2`). However, any other condition may be implemented in the future by deriving an additional class from the base `Condition` class.

Every condition not only has a pointer to a miner who created the condition (the attacker), but also a pointer to one associated action. This follows the condition and action scheme proposed for the construction of mining strategies. The actions proposed in section 3.1 have been implemented as derived classes from an `Action` abstract base class, where every instance of `Action` has a pointer to an associated condition object.

The degree of complexity of the implemented conditions varies. The threshold condition is rather simple and only checks the rank of the miner who specified the condition, as well as the credit balances of some other miner. Even the luck condition is quite simple as it only checks the number of shares the pool has submitted during the current round compared to the number of shares the pool is expected to submit per round. However, the expected value conditions make use of variables such as expected pool duration, all miners' hash rates in the pool, as well as current credit balances. In order to compute expected end of round queue constellations, the `ExpVal2Condition` uses instances of `MinerSortObject`, objects which resemble miner objects and allow for the theoretical estimation of end of round credit balances.

4.4.2 Scenario set up: condition and action

Please note that the selection of these scenarios is not mutually exclusive meaning that the simulation could consist of up to six different attack scenarios being modeled in addition to the normal scenario. Every condition that is created needs an action and every action needs to be aware of its associated condition. Any one of the four actions can be associated with any of the four conditions, however, please note that `hopping` is primarily linked to a `LuckCondition` and vice versa. Should none of these conditions and actions be selected for the simulation then by default a simulation of a mining pool with no attacker will be run. Throughout this section the different actions and conditions will be examined in depth by focusing on theoretical assumptions, as well as certain implementation choices.

Once the mining pool for the normal scenario, or the *base pool*, has been populated, a new mining pool object will be created for every attack scenario that has been selected to be simulated. However, rather than populating these pools with a new set of miners, the newly created pool objects will be populated with the same miner objects as the base pool. Furthermore, all pools receive an ID, a number, where the base pool has ID 0. Essentially, all pool objects will have a vector containing pointers to miners and all miners will have a vector of pointers to the pools they mine in.

In order to set up the different scenarios properly, all mining pool objects apart from the base pool receive a pointer to a `Condition` object that entails a specific `Action` based on their pool ID. For example, a mining pool object that has ID 2 will receive a pointer to a condition that has a pointer to a donation action. Every `condition` is set by an attacker miner and

has a pointer to an **Action** object. Both, the **Condition** and **Action** class are abstract classes. The different type of conditions are:

- **ThreshCondition**: a condition that is based on the attacker being in the top N miners by total credits, as well as a threshold that specifies the number of credits the behind the attacker may not exceed.

The condition is considered active when the attacker is in the top N miners and the subsequent miner's balance are relative to the attacker's credits larger than the specified threshold.

- **LuckCondition**: a condition that makes use of the overall performance of a mining pool during a particular round for the pool by estimating the pool luck, which can be expressed as

$$pool\ luck = \frac{S_E(P)}{S_A(P)} \quad (4.2)$$

where $S_A(P)$ is the actual number of shares submitted for a round by pool P and $S_E(P)$ is the expected number of shares to be submitted per round by pool P .

Furthermore, the luck condition has a pointer to a second mining pool, to which the attacker could, depending on the action associated with the condition, potentially submit shares to rather than continuing to mine for the current pool. The condition is considered active when the pool luck is below the specified *luck limit* for a given round.

- **ExpValCondition**: a condition that uses expected values to compute the expected end of round miner queue constellation.
- **ExpVal2Condition**: a condition that computes and compares an attacker's expected credit differences for N rounds.

As briefly mentioned before, every condition is associated with an action and every action is aware of the condition it is linked to. An action determines the handling of a share submitted by the attacker. Whether an action is pursued or not depends on the state of associated condition. An action can be any of the following types, which are derived from the abstract **Action** class:

- **Share_withholding**: a share is withheld, not submitted to the pool operator. The share does not count towards the shares the miner generated nor his performed work.
- **Donation**: a share is donated to a miner that is specified by the condition. For this report the miner who will receive the donation is the miner that should overtake the attacker in the queue. Unlike share withholding, the work performed by the donating miner, the attacker, is recorded.
- **SecWallet**: a share is donated to the attacker's second wallet. Once a **SecWallet** action object is constructed, a new instance of **Miner** is created and passed into the pool corresponding to the second wallet scenario. This new **Miner** object does not have a hash rate and does not submit any shares but solely receives credits and potential blocks via the work performed by the attacker. It should be noted that although all work performed by the attacker is recorded only one of his two wallets will receive credits. Which of the two wallets should be used strictly depends on the state of the associated condition.

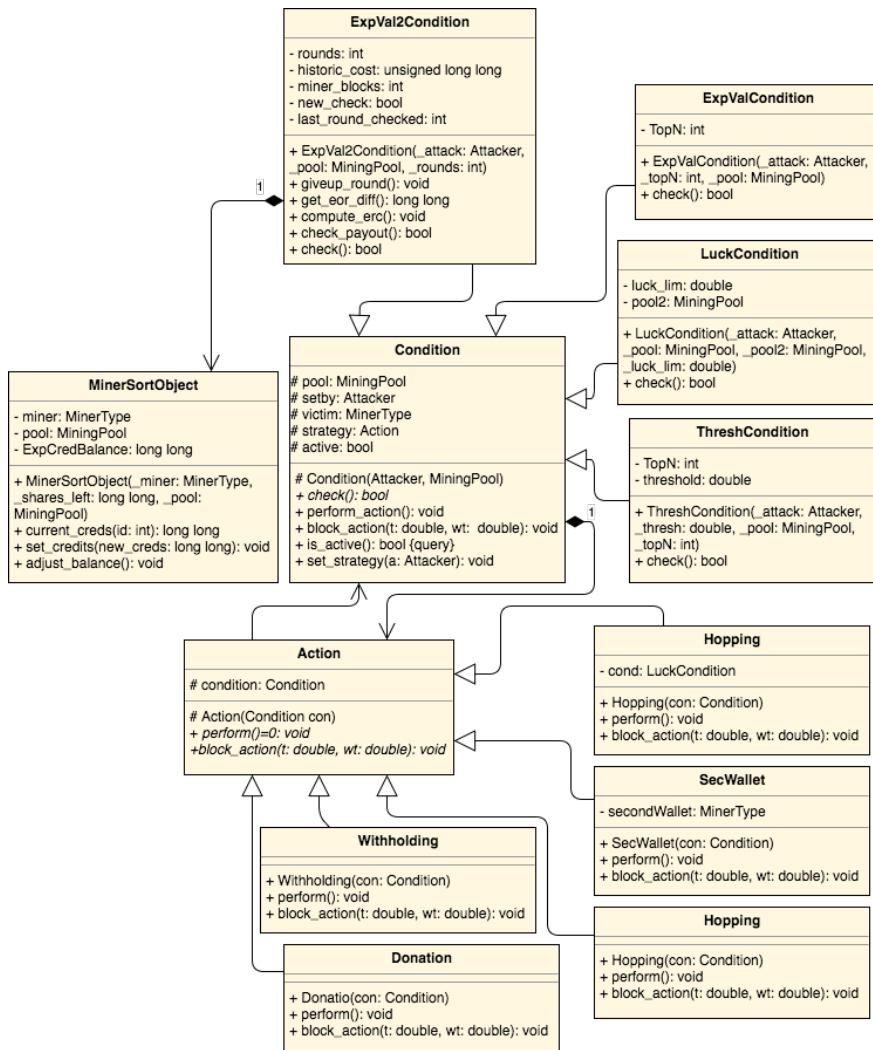


Figure 4.4: A UML diagram exclusively showing the relationship between the different action and condition classes

- **Hopping:** a share is submitted to the pool and the credits are updated. However, instead of scheduling the next share, the attacker first checks whether the linked condition is active. If the condition is active then the attacker will not schedule his next share in the current pool, but rather leave that pool and join a different mining pool that is specified by the condition. Should the condition be inactive or the share be a network share then the attacker remains in the pool he currently mines in and schedules his next share.

4.5 Setting up a simulation

This section outlines the simulation configuration and some of the data collection features offered by EthSim. Furthermore, the construction of mining pools is explained by presenting two possible methods for populating a pool with any number of miners.

4.5.1 The simulation configuration and data collection

The configuration file `Config`, a `json` file that is passed as a command line argument, allows the user to specify a number of settings for the simulation, predominantly the duration of the simulation denoted as number of blocks found by the pool, simulation name, network difficulty, pool name, pool fee, as well as pool share difficulty. Furthermore, the user may choose between populating the mining pool with his own miners by passing in a `.csv` file consisting of miner data, namely miner name and the respective hash rate, or he may populate the pool by sampling for N miners for the hash rate from a log normal distribution. The log normal distribution parameters mean and standard deviation, as well as potential upper and lower bounds may also be specified by the user in the configuration file. In the sampling approach, the name of a miner will also be generated by appending some randomly generated up to eight digit number to the letter `X`. Given the miner population process, either via the passing in of miner data or sampling, the user may also select specific miners to be monitored by specifying their names or via sampling from the pool of generated miners by specifying the percentile the miner's hash rate should lie in, respectively. Miners selected to be monitored will be marked in the simulation and all their actions will be recorded. `EthSim` allows for the collection of various data points relevant for the effective comparison of different mining strategies. A miner object who has been *marked* uses the method `write_data()` to write the time and his credit balance to a specific file each time a share event scheduled by the miner is being executed. This allows for the analysis of the credits development over a period of time for a specific miner.

A mining pool records information by default. Each time a block is found, a network share event is submitted to a particular pool, a mining pool object calls the method `writeCSV()`. Every mining pool object has an output file stream as a member variable and thus there will be a file containing this information for every pool object in the simulation. Using the `writeCSV()` method the mining pool object writes to a `.csv` file the time of the simulation and credit balance of the top miner, the reset balance, the work per block, as well as the luck of the pool for the completed round. By accumulating this data one can visualise the amount of work performed per block relative to the network difficulty, as well as compute average start or win credits, which will be presented throughout the upcoming chapters.

At the end of every simulation two files containing relevant information are created by default. First, a `.csv` containing information, such as blocks won, avg. start credits, avg. win credits, hash rate, number of miners and pool luck, for every mining pool of the simulation is created. Second, a `.json` contains information about all miners that were part of the simulation. The reason for this file to be in `.json` format is due to the existence of nested data for every instance of a miner, such as blocks found, blocks received, total shares submitted, shares donated, share donations received for every mining pool a miner mined in.

Should the user want to simulate a scenario in which there are one or more miners classified as attackers and hence pursue particular actions in certain situations, the configuration file allows for it. Similar to the initial miner populating process for the mining pool, the user can specify the name of the attacker if the pool has been populated with miner data passed in by the user. Alternatively, given a pool of miners that has been generated through sampling, the attacker can be sampled from the pool of miners by specifying the percentile in which the attacker's hash rate should lie.

4.5.2 Populating the pool

Everything specified in the configuration file is processed within the `Simulation` object as soon as the simulation is run. First, the mining pool for the normal scenario is setup by creating a `MiningPool` object and populating it with `Miner` objects according to specifications in the configuration file. A miner object can be constructed by passing in constructor arguments for the name of the miner, the miner's hash rate, as well as a pointer to the mining pool the miner mines in.

In the case that miners have to be generated from sampling, the `Simulation` object makes use of the `MinerGenerator` object every instance of the class has. A `MinerGenerator` object has a method named `createMiners`, which takes amongst other arguments a pointer to a mining pool object that should be populated with miners. As the assumption is that the hash rate of miners in Ethpool follows a log-normal distribution, the `MinerGenerator` object, which is constructed by passing in lower and upper bound limits for the sampling, makes use of the `std::lognormal_distribution` template. This template generates random numbers that are greater than zero according to a log-normal distribution using the probability density function (PDF):

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma x} e^{-\frac{\ln(x)-\mu^2}{2\sigma^2}} \quad (4.3)$$

where μ is the mean and σ the standard deviation [5]. When the `createMiners` method is being called, one of its arguments is the number of miners that should be created, or sample size. Thus, the `MinerGenerator` will sample a hash rate from the distribution, generate a random number for the wallet address, or name, and subsequently create a new `Miner` object and pass a pointer to the newly created miner to the mining pool for a number of times equal to the sample size.

Chapter 5

The *Two Miner Case*

5.1 A simple scenario

This section evaluates potential attack strategies in a two miner case, as initially proposed by Zamyatin et al. [20]. In the two miner case, the assumption is that only two miners, a small one and a large miner mine in the same queue-based mining pool. The large miner, *miner one*, mines at a rate of 1 GH/s¹ and the small miner, *miner two*, at a rate of 100 MH/s. Even though such a scenario may not be an accurate representation of a real world queue-based mining pool, such as Ethpool, it should provide a good overview of the workings of a queue-based pool system, as well as allow for a first comparison of the effectiveness of different attack strategies in this simplified scenario.

5.2 Simulation configuration

For the simulation configuration, the network difficulty has been set to 183.671TH (trillion hashes) and the share difficulty to 3.6 billion hashes. Perfect network connectivity has been assumed for all miners, as well as a constant hash rate. In addition, uncle blocks, as well as stale and invalid shares are not taken into account. The *Attacker* is the 1 GH/s miner m_1 and the *Victim* is the 100 MH/s miner, m_2 . The duration for the two miner case simulation has been set to 100 000 blocks.

The attack strategies being simulated all follow the same condition:

If miner m_1 is ranked top of the queue in terms of his overall credit balance and the credit balance of miner m_2 is within a 90% threshold of m_1 's credits, then the condition is active.

A scenario will be simulated for each of the three considered actions linked to the condition: the withholding of shares, donation of mining power, as well as the use of a second wallet, as explained in section 3.1. A normal scenario will be simulated, whereby the attacker does not pursue any attack strategy, in order to have an appropriate scenario against which the performance of each attack strategy employed can be compared.

¹1 GH/s = 10⁹ hashes per second

5.3 Simulation results and evaluation

Table 5.1 shows the rewards received and work performed by both miners for each of the different scenarios. *Blocks rewarded* refers to the number of blocks a miner has received, or been granted, from the pool operator, i.e. the number of times a miner was top of the queue when a block was found by the pool. *Blocks mined* refers to the number of shares submitted by a miner, which were valid solutions to the network problem, i.e. a block was found by the pool because of the submitted share.

Attack strategy	Miner	Avg. performed work per block (trillion hashes)	Blocks rewarded	Blocks mined	Ratio
None (solo mining)	Attacker	183.683	90898	90898	1.0
	Victim	183.423	9102	9102	1.0
None (normal scenario)	Attacker	185.047	90228	90898	0.9926
	Victim	170.847	9772	9102	1.0736
Share withholding	Attacker	178.427	72973	70886	1.0294
	Victim	237.992	7015	9102	0.7707
Tactical donation of mining power	Attacker	182.892	91291	90898	1.0043
	Victim	191.700	8709	9102	0.9568
Use of a second payout address	Attacker	185.363	90074	90898	0.9909
	Victim	168.196	9926	9102	1.0905

Table 5.1: Results for a simulation of a two miner case for 100 000 blocks

5.3.1 Simulation of a two miner pool

The credit development of miners m_1 and m_2 for the first 100 blocks in the normal scenario is visualised in Figure 5.1. Due to the time scale in Figure 5.1, the credit growth may appear linear at first. However, the figure also shows a small part of the credit development for the 10 GH/s miner m_1 in a subplot with a zoom factor of $\times 3500$. The subplot shows that the time intervals in which valid shares are submitted by a miner are not constant. This should come as no surprise given that finding a valid solution to the pool problem is a Poisson process. It can be seen how the smaller miner m_2 wins a block when his credit balance is close to the network difficulty, whereas the large miner m_1 maintains much larger credit balances, well above the network difficulty.

Figure 5.2 shows the computed hashes per rewarded block for both miner m_1 and m_2 , as well as the correlated luck of the pool, for the first 100 blocks. Pool luck is computed as

$$\text{Pool luck} = \frac{\text{Expected number of shares per block}}{\text{Actual number of shares per block}} \quad (5.1)$$

for every round in a pool.

It can be seen that a miner performs fewer hashes than the network difficulty when the pool is very lucky and vice versa. For example, during the time period between 6 million and 8 million seconds, both miners m_1 and m_2 benefited from lucky rounds and performed significantly less work than the expected amount per block. However, just by looking at

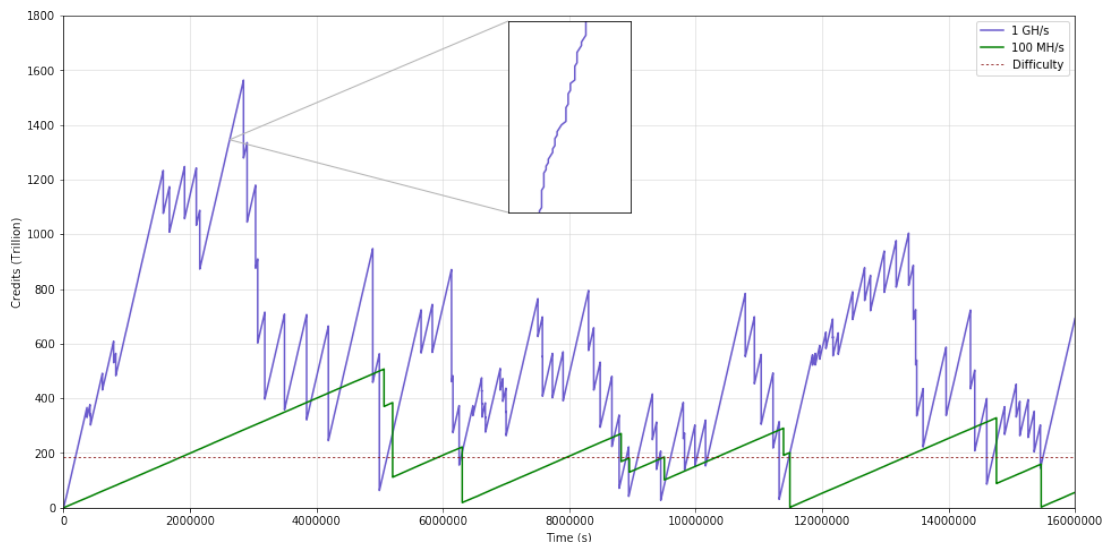


Figure 5.1: The credit development of two miners for the first 100 blocks

the performed work per rewarded block for the first 100 mined blocks, it appears that the large miner m_1 performs a lot more hashes, well above the difficulty, for a block notably more often than the small miner m_2 . In fact, over the course of 100 000 blocks, miner m_1 had to perform on average 14.2TH more per received block than miner m_2 . Furthermore, miner m_1 performed on average 1.376TH more than the network difficulty, whereas miner m_2 surprisingly performed 12.824TH less than the network difficulty. In terms of the rewarded blocks, miner m_1 was rewarded fewer blocks (90 228) than how many he actually mined for the pool (90 898), suggesting that he would have been better off by 670 blocks had he mined solo. Consequently, the small miner m_2 , received 9 772 blocks, a substantially larger amount than how many blocks he actually mined (9 102).

Zamyatin et al. [20] find that the reason for the small miner outperforming the large miner in terms of work per block and blocks received relative to blocks mined is predominantly linked to the attacker being faced with a higher probability than the small miner for reaching the top of the queue during a time when the pool is unlucky, given the significant difference in mining power between miners m_1 and m_2 . More specifically, in such a queue-based scheme, large miners, whose hash rate accounts for a significant amount of the pool’s hash rate, such as miner m_1 , are more likely to absorb large amounts of the mining pools variance caused by lucky and unlucky streaks than smaller miners [20]. This can partially be seen in Figure 5.2, which shows the real work both miners perform per rewarded block, as well as the correlated luck of the pool, for the first 100 blocks.

The credit development for both miners for the share withholding scenario is shown in Figure 5.3, which looks fairly similar to the credit development for both miners for the normal scenario from Figure 5.1. However, even though both figures only account for the first 100 blocks, differences in terms of the credit development for both miners can be made out. In Figure 5.3, the subplot with a zoom factor of $\times 5500$ shows a condition being met multiple times over a short period of time, as miner m_1 stops his submission of shares and only continues to work once miner m_2 surpasses him in the hope that miner m_2 will then win a block and be reset. It can be seen how miners m_1 and m_2 alternate in terms of who is top of the queue almost on a share by share basis. The effective use of the strategy can be

5.3. SIMULATION RESULTS AND EVALUATION

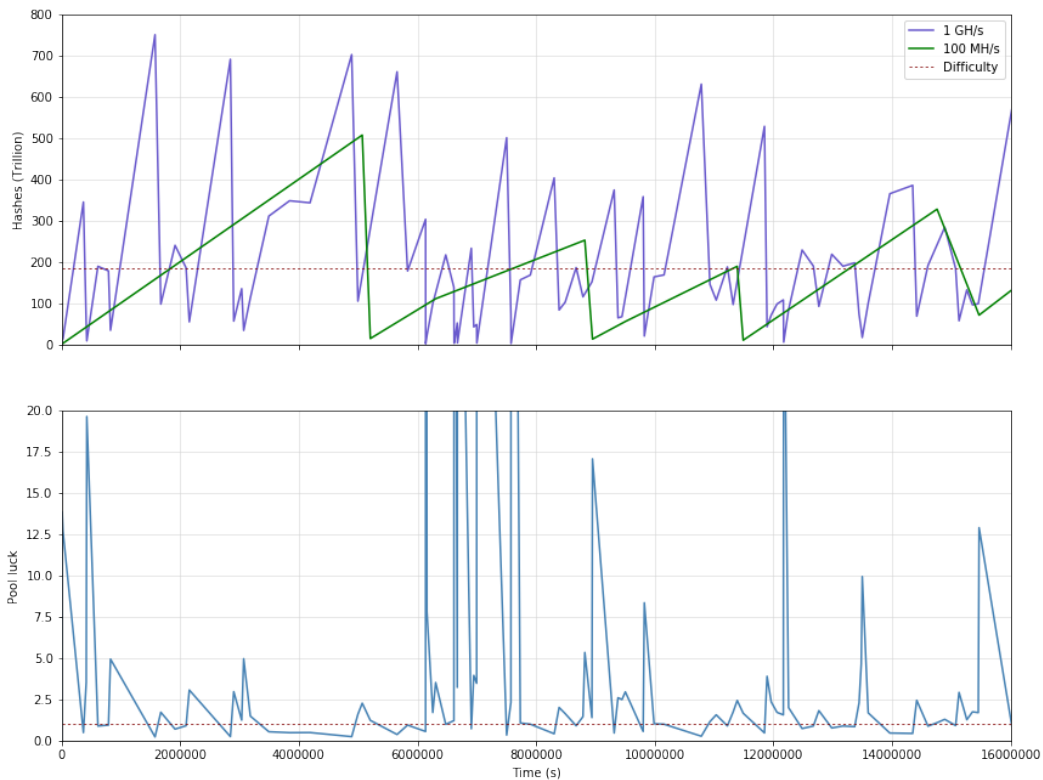


Figure 5.2: The number of hashes computed by two miners per rewarded block correlated with pool luck for the first 100 blocks

seen at around 5 million seconds, where miner m_1 waits to be overtaken by miner m_2 , who subsequently wins the next block and is reset to almost zero credits. Miner m_1 then continues to maximise his mining efforts for a prolonged period of time, while miner m_2 gradually builds up his credits again. Interesting enough, at the same time in the normal case (Figure 5.1) miner m_2 benefits from a lucky round and wins two blocks over a rather short period of time and has his credits reset to an amount significantly larger than zero.

In terms of overall performance and rewards during the share withholding scenario, miner m_1 managed to perform better in terms of his block ratio (1.0294) than had he not pursued an attack strategy (0.9926) or mined solo. Furthermore, compared to the normal scenario, miner m_1 managed to reduce his average work per block by 6.72TH, and thus performed notably fewer hashes than the difficulty. Miner m_2 , however, had to perform 67.145TH more per block than during the network difficulty.

Perhaps share withholding may therefore seem like an ideal reward increasing for miner m_1 , however, the number of blocks found and received by both miners is remarkably lower than for the normal scenario. This is due to all scenarios following the same timeline of events, or shares are being submitted at the same time instances in every scenario. As the share withholding scenario is the only setting in which a share may be ignored, which simulates the real world case for a miner not generating a share at all, it may occur that a valid network share is ignored and consequently no block is found. Taking this block loss into account, even though an attacker like m_1 would be able to reduce his work per block and improve his ratio of blocks received to blocks found, his overall reward would be less for the given time frame relative to the normal scenario, or 17 255 blocks less, as shown by Table 5.1.

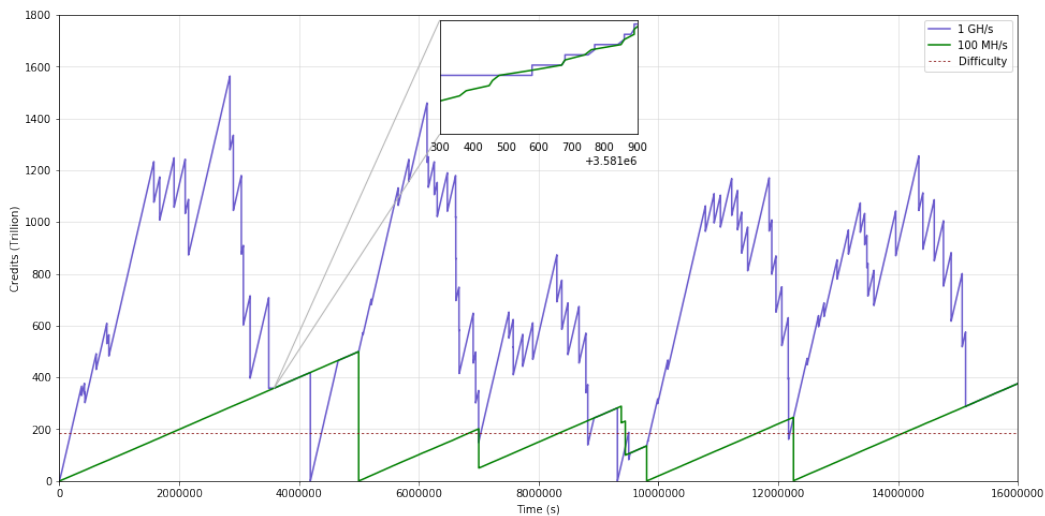


Figure 5.3: The credit development for two miners in a share withholding attack scenario for the first 100 blocks

The use of a second payout address, or wallet, strategy performed worse in terms of performed hashes per rewarded block and number of blocks rewarded for miner m_1 than any other scenario. This may be due to a potential queue constellation in which the wallet of miner m_1 eats into the credit balance of miner m_1 's first wallet [20]. More specifically, creating a second wallet is equivalent to a third miner joining the pool. Therefore, at some point and after several share submissions to the second wallet, it could be the case that miner m_1 's first wallet is top of the queue and the second wallet is ranked second of the queue. Given that the credits of the second wallet are very high relative to the first wallet's. Such a queue constellation could imply that rather than miner m_1 continuing to mine for his first wallet, he submits shares to his second wallet instead, as the specified threshold condition would be met. However, rather than building up his credit buffer to the second ranked miner, he reduces it by submitting shares to his second wallet, which in fact is the second ranked miner. Thus by indirectly causing the diminishing of his end of round credit differences, miner m_1 's second wallet strategy seems to boost the number blocks rewarded to miner m_2 and thereby reduce the average work miner m_2 performed per rewarded block.

The last strategy to analyse is the tactical donation of mining power to the second miner, when the condition is met. For miner m_1 , pursuing this strategy not only proved to be better than the normal scenario in terms of performed work per block and block ratio, but also performed better in terms of the number of blocks rewarded. Miner m_1 received 1063 blocks more than in the normal scenario, as well as 393 blocks more than had he solo mined. Miner m_2 on the other hand received 393 fewer blocks than how many he actually mined for the pool with an average number of hashes per block that lies above the network difficulty.

The results for the tactical donation scenario not only imply that donating mining power is the best out of all three considered strategies, but is also more profitable in terms of blocks received and performed work than pursuing no attack strategies or mining solo. Even though the results from the share withholding scenario may at first suggest that the strategy is better, given that a significant number of blocks is lost relative to the normal scenario, it does not outperform the donation strategy. For a two miner case, it could also be seen that the second wallet approach is not worth to be pursued by a miner, as pursuing such a strategy has a

potential negative effect on a miners rewards, due to the creation of a *third* miner, namely the second wallet.

5.3.2 Problems with the two miner case

Even though the results suggest that large miners can make up for the extra work they have to perform per block, particularly by employing a donation strategy, one can not determine whether such strategies will be viable in the real world case for Ethpool purely based on the outcomes of a two miner case simulation. In fact the two miner case could be misleading as the condition used by the attacker is very naive in the sense that it makes use of only two variables, namely the attacker's position in the queue relative to that of the small miner, and the total credits of the small miner relative to the attacker's credits. In reality Ethpool would be considered a *multi-miner case*, where several additional variables have to be taken into account in order to determine a favorable condition for a miner to pursue an effective attack strategy.

In a multi-miner case an attacker has to be very attentive to the current constellation of the queue when a condition is checked as part of a reward-increasing strategy. For example, in the two miner case, when an attacker gives up a round by allowing the small miner to surpass him in terms of total credits, the large miner is certain to win the next round. However, in a multi-miner case, assuming the attacker is not the largest miner in the pool, it could very well be the case that the attacker gives up a round to some miner, but fails to win the following round as other miners with higher hash rates move up the queue a lot quicker, potentially overtaking the attacker. Even if an attacker is the largest miner in the pool and gives up a round it could be that the end of round credit difference he receives the following round is significantly smaller than the initial difference he could have received. Such a situation could arise from smaller miners with relatively lower hash rates residing close to the top of the queue being overtaken by miners with significantly larger hash rates. Therefore, if an attacker does not define a condition as part of his attack strategy, which accounts for factors such as other miners' hash rates, current credit balances, or the length of the current round, he could actually find himself worse off in terms of his payout than had he not pursued any attack strategy in the first place. How the naive conditions used in the two miner case would perform in a multi-miner case, as well as what kind of conditions would take more variables into account will be the focus of the next chapter.

Chapter 6

The *Multi-Miner Case*

6.1 Simulation configuration

Throughout this section multiple simulations of mining strategies in pools of various sizes will be examined. All simulations discussed were set up using the same configuration settings as the simulation for the two miner pool presented in section 5.2, apart from the simulation duration, which was set to 200 000 blocks for the multi-miner pool simulations.

However, between the multi-miner case and the two miner case simulation setup one other difference is the generation of miners for the simulated mining pools. In the two miner pool simulation, both miners were specified by name and hash rate via a .csv file. For the simulations used throughout this section, the miners have been generated by sampling. Zamyatin et al. [20] obtained the name and hash rate for 729 miners in Ethpool via the public Ethpool API between 21-02-2017 and 09-04-2017. Figure 6.1 shows the distribution of hash rates for the 729 Ethpool miners. It can be seen that the hash rate distribution in Ethpool resembles a log normal distribution. An additional representation of the Ethpool hash rate distribution is provided in appendix A.2. Thus, in order to construct queue-based mining pools of different sizes, sampling for the hash rate of miners using a log normal distribution allows for the unbiased populating of each pool. The probability density (equation 4.5.2) function employed for the sampling used the mean and standard deviation of the hash rate data obtained for the 729 Ethpool miners¹.

This chapter examines the workings and effectiveness of mining strategies employed in any queue-based pool and should thus not be too specific to Ethpool. Therefore, instead of using the data for the 729 Ethpool miners obtained by Zamyatin et al. [20], the standard deviation and mean of the recorded hash rates have been used to create a log normal distribution of hash rates and consequently sample from it in order to generate miners for a pool. Thus for the populating of pools, it is assumed that the hash rate distribution of a queue-based pool is log normal. Given recent increases in the number of miners in Ethpool, as well as the sighting² of very large Ethpool miners, the upper bound for the sampling has been set to 120 GH/s. Therefore, for every simulation a mining pool will be populated by N miners via sampling N hash rate values from the distribution. Once the pool size has been met, the attacker will be determined by sampling a miner from the pool whose hash rate lies within

¹The mean of the 729 hash rates was 0.959 GH and the standard deviation was 1.742

²Ethpool maintains a ranking of the top miners by hash rate, which is shown in Appendix A.1

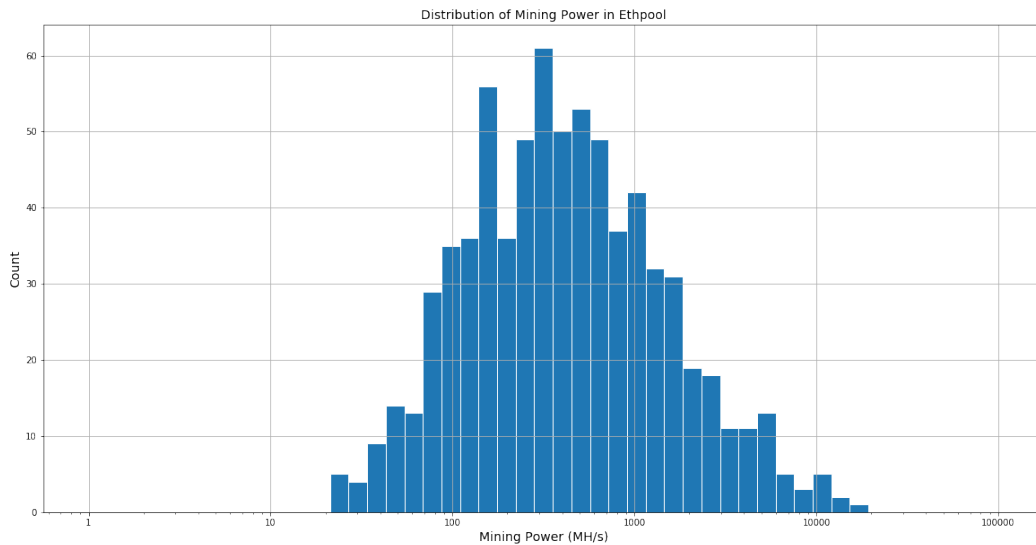


Figure 6.1: The distribution of hash rates of 729 Ethpool miners (logarithmic scale). [20]

a specified percentile. For the simulations presented, an attacker’s hash rate was set to lie within the 80th and 90th percentile of all miners in the pool. The reason for using these particular percentile values is to ensure that the attacking miner should have a relatively high hash rate compared to the other miners in a pool.

6.2 Naive conditions in the multi-miner case

The aim of this section is to analyse the effectiveness of the naive mining strategies presented in the previous chapter in a multi-miner scenario. Specifically, the effectiveness of the condition specified for these mining strategies will be assessed. The condition from the two miner case only took into account the second ranked miner’s credits when the attacking miner was top of the queue.

For the evaluation of these strategies, three different simulations will be evaluated: a pool with 1 000 miners, a 100 miners pool, as well as a relatively small 10 miners case.

The first multi-miner case to be examined is a queue-based mining pool with 1 000 miners. The distribution of hash rates of the 1 000 miners that have been generated is displayed in Figure 6.2 and appears to be, as expected, log normally distributed.

In terms of the pool behaviour over the course of 200 000 blocks, the top plot of Figure 6.3 shows the development of the credits a top miner had when a block was mined by the pool. A miner is expected to win a block when he has performed an amount of hashes equal to the network difficulty. However, the credit balance is not an accurate indicator of how much work a top miner has actually performed prior to winning a block due to the scheme’s underlying credit resetting mechanism. In fact, the effects of the non-uniform credit resetting mechanism are visible in the top plot as the majority of the credit balances of winning miners appear to lie above the network difficulty.

On average, a winning miner had a 184.15 trillion credits, which is approximately 0.48 trillion credits above the network difficulty. The middle plot therefore shows the actual amount of

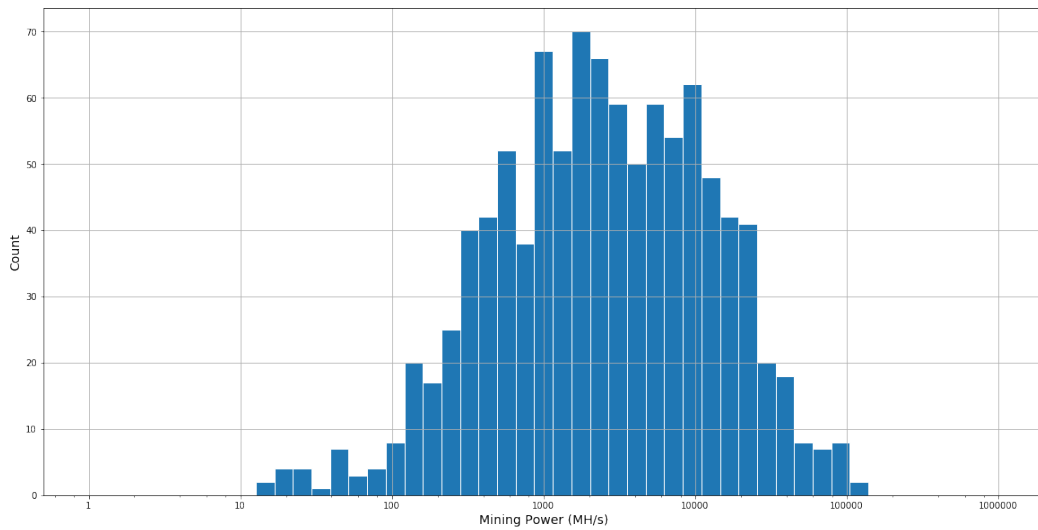


Figure 6.2: The distribution of hash rates of 1,000 miners in the constructed mining pool

work, or number of hashes, a top miner has performed prior to winning a block for each of the 200 000 blocks mined by the pool. Compared to the credit balance of a winning miner, it can be seen that the number of hashes performed by a miner who received a block is in fact closer to the difficulty, the expected amount of work per block. This can also be seen by the performed work of a winning miner, which is on average 183.35 tera hashes (TH), and hence closer to the network difficulty of 183.6 TH than a winning miner’s credits³.

The comparison of credit balances and actual performed work of a winning miner implies that a winning miner could have a higher ratio of credits received per computed hash. The bottom plot of Figure 6.3 shows the actual number of credits received per computed hash by a winning miner. For every block, the credit balance of the winning miner is divided by the number of hashes he performed since the last time he was rewarded a block⁴. It can be seen that some miners benefit from the credit reset mechanism and receive slightly more credits than the amount they actually worked for. The average number of credits received per performed hash by the winning miner was 1.0043 and only few miners actually received a significantly higher amount of credits than the work which they actually contributed. These credit development trends show perhaps what someone would already expect of a simulated queue-based mining pool and is therefore a good indicator for the correct workings of a simulated queue-based pool.

It should be noted that the simulations of the 10, 100, and 1 000 miner pools primarily aim at examining the effectiveness of the proposed mining strategies and will not provide an in-depth comparison between the work performed by large and small miners. However, an in-depth evaluation on performed work in a multi-miner pool of 729 Ethpool miners has been conducted in joint work with Zamyatin et al. [20], where it has been found that larger miners are disadvantaged by the queue-based payout scheme compared to smaller miners in a multi-miner pool. This is shown in Figure 6.4, which displays the ratio of performed work per block relative to the average for the case of Ethpool.

³The average number of hashes per block being slightly less than the network difficulty may be explained by the pool’s luck.

⁴This rests on the assumption that a miner receives one credit per computed hash

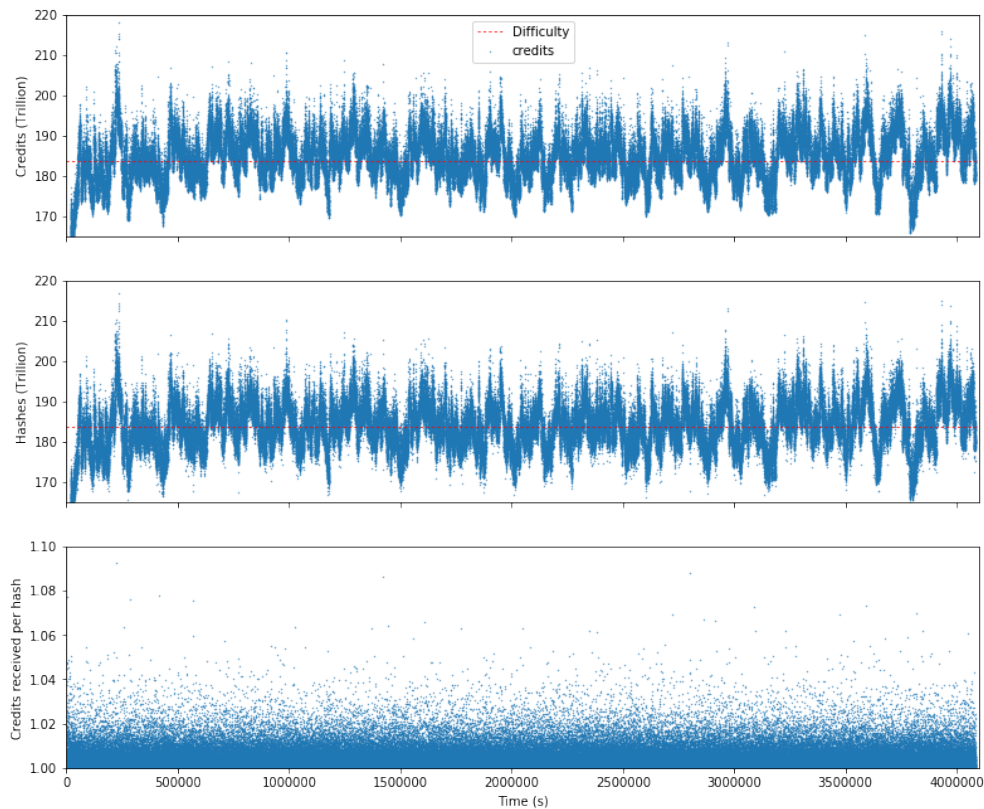


Figure 6.3: Development of the top miners’ credits when winning a block compared to their performed work per block, as well as the ratio of the winning miner’s credits to his performed number of hashes per block in a pool of 1 000 miners for 200 000 blocks.

The condition for the attack actions *share withholding*, *tactical donation of mining power*, and *use of a second payout address* used for the simulations of multi-miner pools was the same as for the two miner pool, presented in section 5.2. The results for each attack strategy pursued in a pool consisting of 1 000 miners are shown in Table 6.1. The sampled attacker had a hash rate of 9.134 GH/s, however, did not outperform the scenario of normal mining in any of the strategic mining scenarios. In fact, using any of the three mining strategies performed slightly worse, as the attacker received one block less (240 blocks opposed to 241). Even though the results for employing the mining strategies are better in terms of average work per block and blocks received than solo mining, the attacker would have been better off pursuing no strategies.

The results for the performed attack strategies in a pool of 100 miners presented in Table 6.2 show that the performance of each attack scenario in terms of blocks received, differs noticeably more than for the 1 000 miner pool. Even though the results still suggest that not pursuing any mining strategy is the most rewarding strategy for a miner (1 790 blocks received for 1 726 blocks mined), the three mining strategies do not all perform equally bad. As for the two miner case, the withholding of shares results in a lower work per block ratio for the attacker, however, at the cost of both, mining and receiving, a fewer number of blocks. Interestingly, the use of a second wallet outperforms the tactical donation strategy by six rewarded blocks. That is potentially due to the likelihood of the attacker’s second wallet eating into the credits of his first wallet being less for a pool with many miners than for a pool with only two.

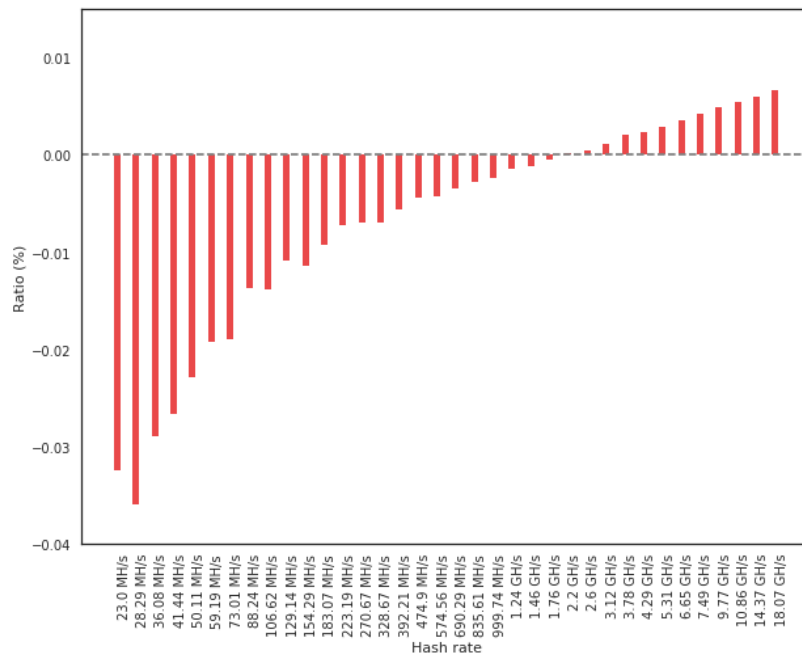


Figure 6.4: The ratio of performed work per block in Ethpool relative to the average amount of performed work per block (miners have been grouped by their hash rate).[20]

For the last simulation of a 10 miner pool, the trend that the normal scenario outperforms the attack scenarios remains, as shown in Table 6.3. However, the scenarios in which attack strategies have been pursued performed significantly worse than normal pool behaviour or solo mining. Interestingly, this time the donation of shares outperformed the use of a second payout address in terms of blocks rewarded, which may be due to the second wallet eating into the attacker’s first wallet’s credits again. Nonetheless, substantial inconsistencies in the performance of the three mining strategies compared to the normal scenario across the three different pool sizes have been found.

A potential reason as to why the strategies perform a lot worse in a 10 miner case than a 1 000 miner case, may be due to the number of times the condition is set active. Across the three different pools, the number of blocks an attacker receives decreases as the pool size becomes larger, which is due to the attacker accounting for a higher proportion of the pool’s total hash rate in a smaller pool than in a big pool. This suggests that the attacker reaches the top of the queue more frequently in a small pool compared to a large pool over the same number of blocks. As the condition for the mining strategies considered so far only checks whether the attacker is top of the queue and the second miner has a relatively high number of credits, the probability of a condition being active in the 10 miner pool may be higher than for the 1 000 miner pool. In fact, for the second wallet strategy, in the 10 miner simulation, the attacker donated 4.068% of his shares, whereas he only donated 0.898% and 0.117% of his shares in the 100 and 1 000 miner simulations, respectively.

The main reason for the performance differences between the different attack strategies employed for the 10, 100, 1 000 miner pool simulations is rooted in the credit resetting mechanism of the payout scheme. As the condition states that a miner has to be top and the second ranked miner’s credit balance has to be within a 90% threshold of the attacker’s credits, the ultimate aim of the three strategies examined is for the attacker to potentially receive

<i>Simulation:</i>	Sim_1000			
<i>Pool size (miners):</i>	1000			
<i>Attacker size (GH/s):</i>	9.134			
Attack Strategy	Avg. performed work per block (TH)	Blocks rewarded	Blocks mined	Ratio
None (solo mining)	190.380	232	232	1.0
None (normal scenario)	183.2707	241	232	1.0388
Share withholding	183.8195	240	232	1.0345
Tactical donation of mining power	184.0304	240	232	1.0345
Use of a second payout address	184.0304	240	232	1.0345

Table 6.1: Results for a simulation of a 1 000 miner pool for a duration of 200 000 blocks

a higher end of round credit difference at some later round by allowing the second miner to overtake him for the current round. The condition does not only ignore the actual improved credit differences and consequent reset balance the attacking miner could earn, but also does not take into account whether it is actually worth for a miner to give up a number of rounds.

Assuming the attacker gives up his round by allowing some miner to overtake him and win the round then a new round starts and 183 Trillion credits are expected to be earned by miners in the pool before the next block is mined. During that time period the attacker may be able to accumulate a very high balance and thereby be reset to an above average number of credits when he wins the round. Alternatively, the queue constellation could change significantly in the sense that some other bigger miner may move up the queue and eat into the credit difference that has been built up by the attacking miner or even overtake the attacking miner and win the next round. The current condition does not take either of these potential outcomes into consideration, as it does not check whether the attacker is expected to receive a higher amount of credits if he gives up the current round, or if he is even expected to win the next round. Thus it may be of no surprise that a mining strategy, which only makes use of a fixed credit threshold and ignores the other miners' hash rates and credit balances, may be ineffective in a queue-based mining pool of more than two miners.

6.3 Expectation-based conditions

The previous section highlighted that the mining strategies examined so far were ineffective due to their underlying condition. The condition was too naive in the sense that it did not account for the hash rates of other miners, the current queue constellation, or the expected end of round credit differences for the current and upcoming rounds.

This section proposes two alternative *expected value-based* conditions used for two new mining strategies: *ExpVal1* and *ExpVal2*. Both strategies make use of expected end of round queue constellations and use these to determine whether an associated action should be pursued or

<i>Simulation:</i>	Sim_100			
<i>Pool size (miners):</i>	100			
<i>Attacker size (GH/s):</i>	6.950			
Attack Strategy	Avg. performed work per block (TH)	Blocks rewarded	Blocks mined	Ratio
None (solo mining)	184.199	1726	1726	1.0
None (normal scenario)	177.613	1790	1726	1.0371
Share withholding	177.4964	1775	1712	1.0368
Tactical donation of mining power	179.0135	1776	1726	1.0290
Use of a second payout address	178.4107	1782	1726	1.0324

Table 6.2: Results for a simulation of a 100 miner pool for a duration of 200 000 blocks

<i>Simulation:</i>	Sim_10			
<i>Pool size (miners):</i>	10			
<i>Attacker size (GH/s):</i>	14.400			
Attack Strategy	Avg. performed work per block (TH)	Blocks rewarded	Blocks mined	Ratio
None (solo mining)	184.683	16 545	16 545	1.0
None (normal scenario)	183.078	16 690	16545	1.0088
Share withholding	182.353	16 044	15 816	1.0144
Tactical donation of mining power	187.574	16 290	16 545	0.9846
Use of a second payout address	190.2954	16 057	16 545	0.9705

Table 6.3: Results for a simulation of a 10 miner pool for a duration of 200 000 blocks

not. The simulation results discussed in this section were obtained from the same simulations of the 1000, 100, and 10 miner pools discussed in the previous section. Therefore it is fair to compare the effectiveness of the previously examined strategies to the performance results of the new strategies.

6.3.1 Expected end of round queue constellation

In the real world case of Ethpool expected end of round credit balances can be calculated for a miner, if one knows the miner's current credit balance and hash rate, assuming the later remains constant. The expected time it takes Ethpool to find a block is always equal to $\frac{D}{H}$ seconds, where D is the network difficulty and H the pool's overall hash rate per second.

In order to compute the expected end of round queue constellation, one can calculate the expected number of credits remaining $E[c_{remain}]_r$ for a round r . The expected number of outstanding shares $E(S_o)$ for round r can be computed as

$$E[S_o]_r = \frac{D}{d} - S_{s_r} \quad (6.1)$$

where $\frac{D}{d}$ is the expected number of shares to be submitted by the pool to find a block and S_{s_r} is the total number of shares submitted by the pool so far in round r . The expected number remaining of credits for round r is thus equal to

$$E[c(remain)]_r = E[S_o]_r d \quad (6.2)$$

assuming that the credits rewarded per submitted share is equal to the share difficulty d . Hence in order to compute the expected end of round credit balance $E[c(m_{(i)})]_r$ for some miner m_i one can calculate

$$E[c(m_{(i)})]_r = \frac{h(m_{(i)})}{H} E[(c_{remain})]_r + c(m_{(i)})_r \quad (6.3)$$

where $c(m_{(i)})_r$ is the current credit balance of miner m_i in round r .

Equation 6.3 allows for the estimation of the end of round credit balances for all N miners in the pool. Ranking these miners based on their credit balances in descending order gives the expected end of round queue constellation for a given moment in time. The calculation for the expected end of round credit balance for some pool miner m_i remains unaltered as the amount of credits a miner will receive from the outstanding credits for the given round is proportional to the miner's hash rate relative to the total hash rate of the pool (equation 6.3). It should be noted that these computations are using expected values and thus there is no guarantee for an outcome.

6.3.2 Expectation-based condition 1: *ExpVal1*

The first expectation-based mining strategy that is proposed, makes use of end of round queue constellations in order to prevent the attacker from performing more work than is

necessary for him to win a round and receive a block. For this new strategy, which shall be called *ExpVal1* strategy, it is checked whether the attacker is in the top N miners in terms of total credits out of all miners in the pool. Given that is the case, the expected end of queue constellation for the current round is computed and it is checked whether the attacker is expected to win the current round. If the attacker miner m_A is expected to win the current round then his expected end of round credit balance is compared to the expected end of round credit balance of miner m_2 , the miner who is expected to accumulate the second most credits by the end of the round. Should the current credit balance of miner m_1 be greater than the expected end of round credit balance of the expected miner m_2 then it is expected that the attacker does not have to do any more work in order to win this round and finish top of the queue. If that is the expected outcome then the condition is set active.

We can thus define the condition as follows:

If the attacker, miner m_A , has the highest accumulated credit balance out of all miners in the pool, then the expected end of round queue constellation should be computed given the other miners' current credit balances, hash rates, as well as the expected number of credits outstanding for the current round. If miner m_A is expected to be top of the queue by the end of the round and the current credit balance of the attacker exceeds the expected end of round credit balance of the miner m_2 that is expected to accumulate the second highest credit balance, then the condition shall be set active.

Instead of defining three different strategies, one for each of the three actions previously examined, the action that is linked to the *ExpVal1* condition is the use of a second payout address. The motivation for the strategy's underlying condition is that when an attacking miner is expected to win the current round and has already accumulated more credits than the expected end of round credits of the miner with the second highest balance, he is already expected to win the round. Therefore, rather than building up his buffer, he will submit shares to his second wallet until the condition becomes inactive. Perhaps the attacker maximising his work efforts for his first wallet and thereby building up a credit buffer may seem like the more appropriate behaviour at first. However, if the end of round constellation is different to the expected constellation, or the pool is unlucky, it could be that a bigger miner is eating into the attacker's accumulated credit buffer. As the idea for Ethpool's credit resetting mechanism is to reward top miners for the extra work they do, missing out on credits due to a big miner eating into the accumulated credit buffer of the top miner may be a scenario worth avoiding. Submitting shares to a second wallet is therefore an appropriate action an attacker may take in order to maximise his work efforts by building up a second wallet, rather than focusing on maximising the credit difference to the second ranked miner.

It should be noted that the *ExpVal1* strategy has been implemented in a way such that when a pool is unlucky during the current round and hence the actual number of shares submitted exceeds the expected number of shares to be submitted by the pool to find a block, the condition is set inactive⁵. Each time the condition is set inactive, the attacking miner submits his share normally to his first wallet.

Simulation	Attack Strategy	Avg. performed work per block (TH)	Blocks rewarded	Blocks mined	Ratio
<i>Sim_10</i>	None (normal)	183.078	16 690	16 545	1.0088
	ExpVal1	189.941	16 087	16 545	0.9723
<i>Sim_100</i>	None (normal)	177.613	1 790	1 726	1.0371
	ExpVal1	178.011	1 786	1 726	1.0348
<i>Sim_1000</i>	None (normal)	183.270	241	232	1.0388
	ExpVal1	184.0343	240	232	1.0345

Table 6.4: The results of the ExpVal1 strategy used in a 10, 100 and 1000 miner pool compared to the rewards received when no strategy is pursued in a simulation of 200 000 blocks.

6.3.3 ExpVal1: simulation results

The performance of the ExpVal1 mining strategy compared to the case of not pursuing any mining strategy is shown in Table 6.4. The strategy does outperform the case of solo mining yet fails to perform better than mining in the pool normally. Even though the attacker is not giving away any of his credits by donating shares to other miners, he may suffer from the second wallet, once again, eating into his first wallet’s credits, especially in a pool of a smaller size, as he is more likely to be top of the queue. The condition does make use of expected end of round queue constellations, and thereby takes into account variables specific to a queue-based pool with more than two miners, yet, it focuses on not performing any extra work rather than taking advantage of above average reset balances.

6.3.4 Expectation-based condition 2: *ExpVal2*

The second proposed expectation-based strategy, *ExpVal2*, makes use of a new condition that aims to identify, for a given queue constellation whether it is worth for an attacker to give up the round or not. The condition for ExpVal2 is similar to the condition proposed for ExpVal1 in the sense that it also computes expected end of round queue constellations. However, the idea for the condition is that it is only worth for a miner to give up a round he could win if the credit payout, the end of round credit difference he could receive by winning a later round, is greater than the credit cost. The term *credit payout* for this strategy refers to the number of credits a miner could receive by giving up N rounds relative to the total cost of credits. The *credit cost* for giving up N rounds refers to the maximum number of credits a miner could have had by winning an earlier round and resume mining until round N .

For the ExpVal2 strategy, the condition entails several aspects. First, the expected end of round credit balances for all pool miners are computed. It is checked whether the attacker is expected to be top of the queue by the end of the current round, and thereby expected to receive the next block reward. Instead of computing the expected credit balances for all miners in the pool, the balances are only calculated for the miners whose current credit balance is greater than the attacker’s credit balance less than the expected number of credits remaining for the given round. If the expected number of credits remaining is greater than the attacker’s current credit balance, then the expected end of round credit balance will be computed for all miners in the pool.

⁵This is due to expected end of round queue constellations not being able to be computed, as there is no expected number of credits remaining

If the attacker is expected to win a round, he has two options. He could either continue to submit his shares normally as he is expected to win the round or he could pursue an action, i.e. donate to a second wallet, and let someone else win the round. The reason for why a miner might want to let some other miner "steal" his round is due to the non-uniform end of round credit differences between the first and second ranked miners. If the attacking miner estimates that he would still be in a favourable position to win the succeeding round given that he intentionally does not win the current round, then he might anticipate a larger difference in credits and thus a better resetting position.

Whether an attacker should deliberately not win a pool round depends on the upcoming expected end of round credit differences between the top two miners for the next N rounds. More specifically, once the attacker miner has checked whether he is expected to win the current round he can also check the expected end of round credit difference for the first and second ranked miner, himself being the former. Thus the expected end of round credit difference between the first and second ranked miners m_1 and m_2 , respectively, is

$$E[c(m_1)] - E[c(m_2)] \quad (6.4)$$

where $c(m_1)$ is the expected credit balance of the miner that is expected to have the highest credit balance of all miners and $c(m_2)$ the expected end of round credit balance of the miner that is expected to have the second most accumulated credits in the pool.

For a miner to *give up a round* suggests that a miner, who is expected to have the highest credit balance by the end of a round, deliberately chooses not to win the current round in the pool by allowing some other miner to accumulate a higher credit balance. A miner can do this by computing the expected end of round queue constellation and checking the expected credit difference between the miners that are expected to be ranked first, m_1 , and second, m_2 , in terms of credits. Thus, miner m_1 could give up a round by not submitting shares accounting for a total of $c(m_1) - c(m_2)$ credits to his main wallet. How to deal with the shares that should not be submitted depends on the action taken by miner m_1 , i.e. donate shares to a second wallet until the attacking miner is no longer expected to win the current round.

In order to be able to compare the expected credit differences a miner could be reset to after he gives up N rounds, the expected cost of deliberately giving up a round, or waiting a round, has to be determined. Were a miner to give up one round r_1 , assuming he would win the following round, then the end of round credit difference of r_2 should be greater than the initial expected end of round credit difference for r_1 . However, this implies that a miner could give up r_1 , maximise his work throughout r_2 , only to end up winning r_2 and receive a slightly larger amount of credits. This may seem a bit unfair given that had he won r_1 , he would have already started working his way up from the bottom of the queue again by the time r_2 finishes.

As Ethpool rewards one credit per computed hash the expected number of credits given to miners per round is equal to D , the network difficulty. Any miner m_j in the pool can thus expect to receive $\frac{h(m_j)}{H}D$ credits for a round, assuming that the hash rate for some miner m_j , $h(m_j)$, and the pool hash rate H remain constant. Therefore an approach towards estimating the expected credit payout of an attacker accounting for giving up i rounds is

$$E[P]_{r_i} = E[\text{erd}]_{r_i} - (\max(E[\text{erd}]_{r_{i-1}}, E[\text{cgr}]_{r_{i-1}})) \quad (6.5)$$

$$E[\text{erd}]_{r_i} = \begin{cases} E[c(m_A)] - E[c(m_2)], & \text{if } E[c(m_A)] > E[c(m_j)] \forall j \\ 0, & \text{otherwise} \end{cases} \quad (6.6)$$

$$E[\text{wpr}(m_A)] = \frac{h(m_A)}{H} D \quad (6.7)$$

$$E[\text{cgr}]_{r_i} = \max(E[\text{erd}]_{r_{i-1}}, E[\text{cgr}]_{r_{i-1}}) + E[\text{wpr}(m_A)] \quad (6.8)$$

$$E[\text{cgr}]_{r_0} = 0 \quad (6.9)$$

where $E[P]_{r_i}$ is the expected credit payout and $E[\text{erd}]_{r_i}$ is the expected end of round credit difference an attacking miner m_A could receive after giving up i rounds. The expected credit payout after waiting i rounds is $E[c(m_A)]_{r_1} - E[c(m_2)]_{r_i}$, where $E[c(m_A)]_{r_1}$ is the expected end of round credit balance of the attacker and $E[c(m_2)]_{r_i}$ the expected end of round credits of the second ranked miner. If miner m_A is not expected to be top of the queue by the end of round i , the expected end of round difference is 0. The expected cost of giving up i rounds is subtracted from the expected end of round difference for round i . This cost is computed by comparing the amount of credits miner m_A has lost by giving up a round $E[\text{cgr}]_{r_i}$ ⁶ (the expected end of round difference) and the number of credits miner m_A could have had if he won the previous round r_{i-1} and resumed normal mining for round r_i , thus receiving the expected number of credits for his work per round $E[\text{wpr}(m_A)]$ ⁷. The larger of these two credit values should be carried forward as the expected cost of giving up r_{i+1} rounds. Only if the expected payout after giving up N rounds is greater than zero, it is worth for a miner not to win the current round in order to benefit from a larger end of round credit difference in a later round.

6.3.5 ExpVal2: a hypothetical example

Perhaps it is worth examining a hypothetical example in order to fully understand the motivations for the ExpVal2 strategy. Table 6.5 shows for a given instance in a round r_0 (current round) the expected credit payouts a miner, m_A , computes for giving up a maximum of 5 rounds. It is assumed that miner m_A receives 5 credits for the work he performs per round. The miner would compute these expected payouts in order to determine whether it is worth for him to pursue a particular action or not, or whether the condition is met or not.

The table shows that miner m_A is expected to win the current round and have his credit balance reset to 10 credits, assuming he has not given up any rounds yet. By computing the expected end of round differences between the top two miners for the following 5 rounds it can be seen that the end of round difference is greater than zero, suggesting that miner m_A is expected to win each of the rounds. If miner m_A would give up, for example, round r_0 and not be expected to win the following round r_1 , the expected end of round credit difference for miner m_A would be zero for the respective round. By looking at the expected end of round credit difference miner m_A could be reset to if he does not win round r_0 , it can be seen that the expected new end of round difference for round r_1 is 22 credits. However, one has to bear

⁶ cgr = cost of giving up round

⁷ wpr = work per round (denoted in credits received for a round)

Rounds Waited	0 (current)	1	2	3	4	5
Exp. end of round difference	10	22	30	36	32	38
Cost of giving up round	(0)	(15)	(20)	(25)	(30)	(35)
	0	+7	(27)	(32)	(37)	(42)
			+3	(35)	(40)	(45)
Exp. credit payout				+1	(41)	(46)
					-9	(37)
						-8

Table 6.5: A theoretical example of computing the expected end of round credit payout relative to the cost of giving up a number of rounds

in mind that miner m_A has an expected cost for giving up one round, namely the number of credits he is expected to receive for one round worth of his work. In addition, he misses out on the initial credit difference he could have been reset to after winning round r_0 , namely 10 credits. Therefore, miner m_A should only give up round r_0 for round r_1 if the expected credits miner m_A would be reset to after winning round r_1 (22 credits) is greater than the expected number of credits miner m_A would have received for the round he has given up, $\frac{h(m_A)}{H}D$, or 5 credits. Additionally, the expected difference which miner m_A missed out on by giving up round r_0 (10 credits) should be taken into account, giving a total expected cost of 15 credits. Therefore, when only taking into account the next round, miner m_A would in fact be better off giving up one round.

It becomes more interesting when looking at the expected credit payouts for giving up 2 to 5 rounds. Miner m_A is expected to win a round after giving up the next 5 rounds and have his balance reset to 38 credits. If one accounts for the 5 rounds of work he has lost, as well as the number of credits he could have been reset to by winning round r_0 , he would face an expected cost of 35 credits, and hence benefit from an additional 3 credits, suggesting it is worth to give up the next 5 rounds. Even though it is correct to account for the work lost for every round given up, the initial end of round difference given up should not necessarily be taken from round r_0 . In fact, not only the initial end of round difference for r_0 should be carried forward. For every additional round a miner gives up r_{0+i} , the expected end of round credits for round r_{0+i} should also be carried forward as an expected cost for any additional round given up. For example, giving up 2 rounds could leave miner m_A with an expected credit difference of 30 credits after winning round r_2 . However, there are two costs to consider. First, the cost of winning round r_0 plus the expected credits miner m_A could have received from mining normally until round r_2 . The second cost would be the expected cost of giving up one round r_0 , winning round r_1 and continuing to mine normally for round r_2 . The example shows that miner m_A could have had an expected amount of 20 credits had he won round r_0 and mined for two rounds normally, or, alternatively 27 credits had he given up round r_0 , won round r_1 and mined for round r_2 . Therefore, using the highest expected cost, it can be seen that miner m_A 's benefit would be an extra 3 credits and hence it would be favourable for him to give up the next two rounds. For determining expected costs, always the highest accumulated cost should be taken forward and compared to new cost incurred from giving up the following round. A good example can be seen when miner m_A estimates the cost for giving up 5 rounds, which shows that the maximum expected cost

would be of giving up rounds r_0 , r_1 , r_2 , winning round r_3 and mining for rounds r_4 and r_5 , or 46 credits.

6.3.6 ExpVal2: simulation results

The theoretical example highlighted that accounting for various scenarios, in terms of credit costs, is essential in order to determine whether it is worth for a miner to give up a certain number of rounds or not. For the implemented ExpVal2 condition in EthSim, the historic cost, namely the number of rounds a miner has already given up, is also taken into account. Thereby each time a miner is about to schedule the next share, he computes the expected payouts for the next N rounds taking into account the cost incurred by the number of rounds he has already given up. As soon as he wins a round, the historic cost is reset to zero.

The performance of the ExpVal2 strategy applied in the same 10, 100 and 1000 miner simulations used for the evaluation of the previous strategies are given by Table 6.6. For the 10 miner pool simulation, the ExpVal2 strategy outperformed the normal scenario by an impressive 390 blocks. Given that a miner receives 5 ETH per rewarded block (minus the pool fee), the attacker would have made an extra \$698,860⁸ more from pursuing the ExpVal2 strategy. As the attacker would have already received more blocks by mining normally in the pool opposed to by himself, the ExpVal2 strategy thereby also performs better than the solo mining case by 535 blocks, and hence outperforms all the naive strategies, as well as the ExpVal1 strategy, which performed worse than solo mining as presented in sections 6.2 and 6.3.2, respectively.

Simulation	Attack Strategy	Avg. performed work per block (TH)	Blocks rewarded	Blocks mined	Ratio
<i>Sim_10</i>	None (normal)	183.078	16 690	16 545	1.0088
	ExpVal2	178.898	17 080	16 545	1.0323
<i>Sim_100</i>	None (normal)	177.613	1 790	1 726	1.0371
	ExpVal2	176.922	1 797	1 726	1.0411
<i>Sim_1000</i>	None (normal)	183.271	241	232	1.0388
	ExpVal2	183.271	241	232	1.0388

Table 6.6: The results of the ExpVal2 strategy used in a 10, 100 and 1 000 miner pool compared to the rewards received when no strategy is pursued over a period of 200 000 blocks.

For the case of a 100 miner pool, the Expval2 strategy once more outperformed all other strategies presented so far. The number of blocks the attacker was rewarded in the ExpVal2 scenario exceeds the number he received in the normal scenario by 7 blocks, a significantly smaller difference than for the 10 miner pool. Perhaps even more interesting, for the case of 1000 miners, the ExpVal2 strategy performed equally well as not pursuing any mining strategy. Before potential reasons are discussed as to why the effectiveness of the ExpVal2 strategy differs so much across pools of different sizes, a last strategy based on pool luck rather than credit differences is examined.

⁸Using the price of \$362.01 per Ether as of 29-08-2017, obtained from <https://coinmarketcap.com/>

6.4 Luck-based condition

So far the majority of attack strategies discussed aimed at increasing rewards for a miner by exploiting the credit resetting mechanism of a queue-based pool. This section proposes a different and more hypothetical approach rooted in the luck of a mining pool.

6.4.1 Hopping: a theoretical approach

Currently, Ethpool is the only Ethereum mining pool that uses a queue-based reward payout scheme. Therefore, in this section the hypothetical scenario in which a second mining pool exists that also employs a queue-based reward payout scheme will be examined. More specifically, rather than trying to propose a condition based on the current or expected queue constellations, a simple luck condition should be examined and linked with a hopping action. Hence, an attacker checks the luck of the pool each time before he schedules a share and to see whether it is below a specified limit. Should the pool in which the attacker mines be very unlucky and reach the specified *luck limit* then the condition will be active and the attacker will leave the pool and start submitting his shares to the second pool. The luck limit is expressed as the expected number of shares a pool has to submit to mine a block relative to a specified maximum number of actual shares submitted by a pool for a given round.

The luck condition has been set to:

If an attacking miner is about to schedule the next share for the pool he mines in, and the current luck of the pool is greater or equal to the specified luck limit, the condition is set active.

The action that is associated to this luck-based condition is the hopping action presented in section 3.1, the strategy shall simply be referred to as *hopping* strategy.

6.4.2 Simulation set up

The simulation configuration is identical to that of the previous multi-miner case configurations. However, one additional mining pool is created and populated with miners that are also sampled from the same log normal distribution of hash rates as for the first pool. For every simulation, the pool sizes for pool 1 and pool 2 have been set to be identical. Furthermore, it is assumed that there is no network lag a miner would face by changing the pool he mines in. A luck limit of 3.5 has been selected for the condition to be active. That would mean that for a given round r in some pool p_i the number of shares that have been submitted are 3.5 times as many as the number of shares the pool is expected to submit to mine a block.

6.4.3 Hopping: simulation results

The simulation results for hopping strategy pursued in a 10, 100 and 1 000 miner pool compared to the miner's performance of not using a strategy are presented by Table 6.7. In the simulation of the 10 miner pool, the hopping strategy performed noticeably better than any of the other strategies previously evaluated. Hopping between two queue-based pools resulted in the attacker receiving 2,301 blocks more than had he mined solo. The attacker

6.5. EXPONENTIAL DIFFICULTY

also received 2,156 blocks more than had he not pursued a mining strategy. However, even though the results for the 100 miner pool simulation still show that using the hopping strategy was the more rewarding scenario, the extra number of blocks won relative to not using a mining strategy diminishes to 1 block for the simulation of the 1 000 miner pool.

Simulation	Attack Strategy	Avg. performed work per block (TH)	Blocks rewarded	Blocks mined	Ratio
Sim_10	None (normal)	183.078	16 690	16 545	1.0088
	Hopping	169.598	18 846	16 545	1.1391
Sim_100	None (normal)	177.613	1 790	1 726	1.0371
	Hopping	172.221	1 847	1 726	1.0701
Sim_1000	None (normal)	183.270	241	232	1.0388
	Hopping	183.497	242	232	1.0431

Table 6.7: The results of the Hopping strategy used in a 10, 100 and 1 000 miner pool compared to the rewards received when no strategy is pursued over a period of 200 000 blocks.

The luck-based condition which has been specified for the hopping strategy may also be naive in the sense that it only considers the luck of the pool the attacker currently mines for, rather than checking the luck of both pools at all times. Furthermore, it does not check any end of round credit balances. In fact, given that only the pool luck of the current round is taken into account, whether an attacker actually benefits from being in a different pool for any later rounds can not be predetermined. Yet, the effectiveness of the strategy varies significantly across the different pools and performs very well for the small pool. The reason as to why the effectiveness of this strategy varies so much for queue-based pools of different sizes may be rooted in some other variable than luck after all.

6.5 Exponential difficulty

Throughout all simulations, it has been assumed that the network difficulty stays constant at 183TH. However, in reality the difficulty may adjust after each mined block. In fact, substantial increases in the Ethereum network difficulty have been observed and are displayed in Figure 6.5. Between March and September 2017, the difficulty rose from 183TH to 2305TH, an increase by 1159.56% [2].

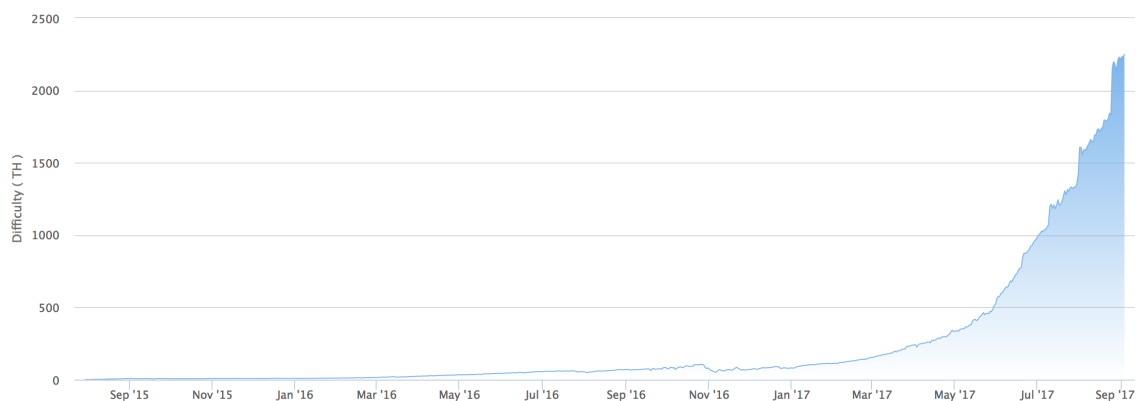


Figure 6.5: Screenshot: Ethereum block difficulty growth chart [2]

The growth resembles exponential growth at an approximate rate of 3.0356×10^{-6} . The start and end blocks used to compute this rate are block number 3356815 and 4233747 for difficulties 183.657TH and 2305.356TH, respectively. Zamyatin et al. [20] have accounted for exponentially increasing PoW difficulty for the two miner and multi-miner cases. However, results suggested that large miners remain the same for a two miner case. Furthermore, large miners remain disadvantaged in a multi-miner pool but at a vaguely smaller scale. Hence it can be assumed that for all simulated mining pools the performance of large miners in the normal scenario should not deviate significantly under exponential difficulty. Additionally, it can be assumed that the effectiveness of the considered mining strategies should not differ significantly under exponential difficulty from constant difficulty, apart from an increase in the performed work.

6.6 Evaluation of queue-based mining strategies

Throughout this chapter several different mining strategies have been proposed and their simulated performance examined. Simulations of three different pool sizes have shown that the naive strategies, which focus on a credit threshold and fixed queue position as successfully utilised in a two miner pool (section 5.3), turn out to perform rather poorly in pools of 10, 100 and 1 000 miners. In fact, pursuing no mining strategy proved to be more rewarding than using any of the naive strategies for each of the three different simulations.

The ExpVal1 strategy did not result in significantly better performance than the naive strategies. The aim of the strategy was to use the expected end of round queue constellation to make an anticipative mining decision and to avoid building up a large credit buffer relative to the miner with the second most credits. Instead the attacking miner would build up a second wallet. Yet, again, not pursuing any attack strategy performed better in each of the three pool simulations.

The ExpVal2 strategy was different to all the other considered strategies, as it focused on identifying favourable queue constellations based on the other miners' hash rates, the queue constellation, and the round duration, as well as the expected cost of giving up rounds, in order to determine the viability of giving up one or more rounds. Employing the ExpVal2 strategy significantly outperformed normal mining or solo in both, the 10 and the 100 miner simulations, but performed equally well as pursuing no mining strategy in the 1 000 miner simulation.

A hopping strategy was proposed as an alternative focusing on the luck of the pool rather than on expected end of round credit differences between the two miners with the highest number of credits. The strategy itself was also rather naive as it did not account for luck of the other pool, which a miner could potentially submit his shares to. Furthermore, the luck of one round does not allow for any predictions about the luck of future rounds as the mining process is *memoryless* due to the underlying Poisson process. However, the results were slightly better than for the ExpVal2 strategy, but the strategy's effectiveness also diminished as the pool size increased.

There have been noticeable inconsistencies in the effectiveness of the considered mining strategies across queue-based mining pools of different sizes. Initially it was proposed, given that the hash rate distribution in EthPool resembles a log normal distribution, simulated mining pools of various sizes should be constructed via sampling from a log normal distribution for

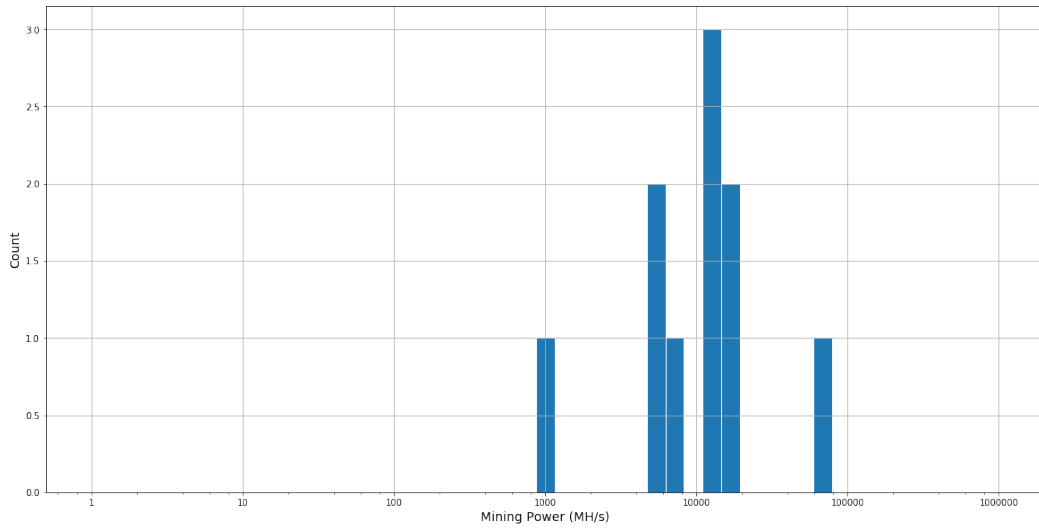


Figure 6.6: The distribution of mining power in a constructed pool of 10 miners (logarithmic scale).

the hash rates. The underlying motivation for such an approach is rooted in the need to examine mining strategies and the workings of queue-based pools of different sizes. Sampling hash rates from a log normal distribution, which used parameters obtained from the Eth-pool data set of miners, allowed for the unbiased populating of queue-based mining pools. However, a 10 miner pool’s hash rate distribution may deviate from the log normal sampling distribution due to the small sample size. As observed in Figures 6.6 and 6.7, the sample distributions for the constructed 10 and 100 miner pools are far from resembling a log normal distribution.

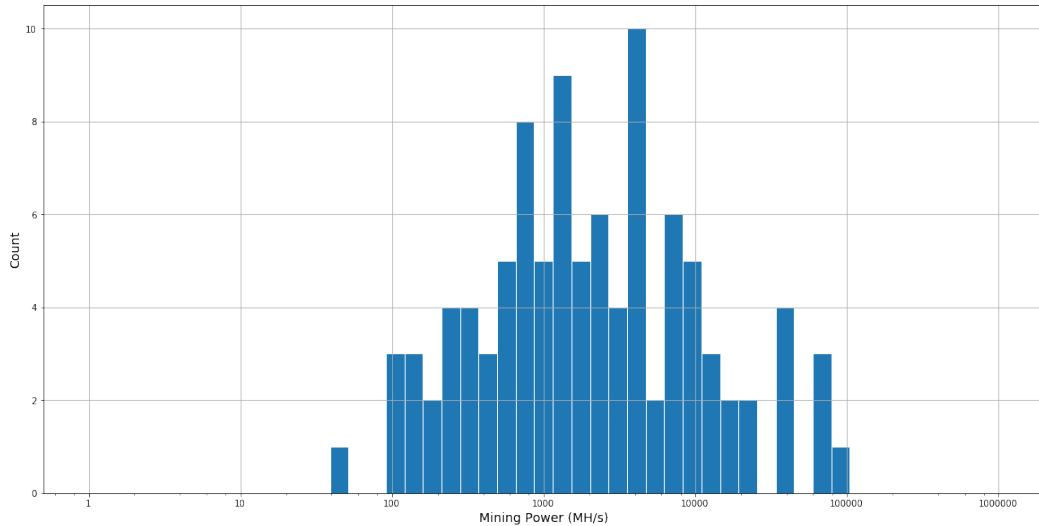


Figure 6.7: The distribution of mining power in a constructed pool of 100 miners (logarithmic scale).

Even for a pool of 100 miners, the sample size may not be sufficient to generate a pool whose hash rates resemble a log normal distribution. In fact comparing these three different mining pools shows that they vary significantly in terms of the average hash rate and hash rate

variance of the miners in each pool. In the 10 miner pool, the average hash rate of a miner was 17.34 GH/s, whereas for the 100 and 1 000 miner pools the average hash rates were 8.03 GH/s and 7.60 GH/s. The hash rate variance was also significantly higher for the 10 miner pool (432.38 trillion) than for the 100 (290.85 trillion) and 1 000 miner (188.3 trillion) pools, potentially highlighting larger differences between the hash rates of miners in the small pool than for the others.

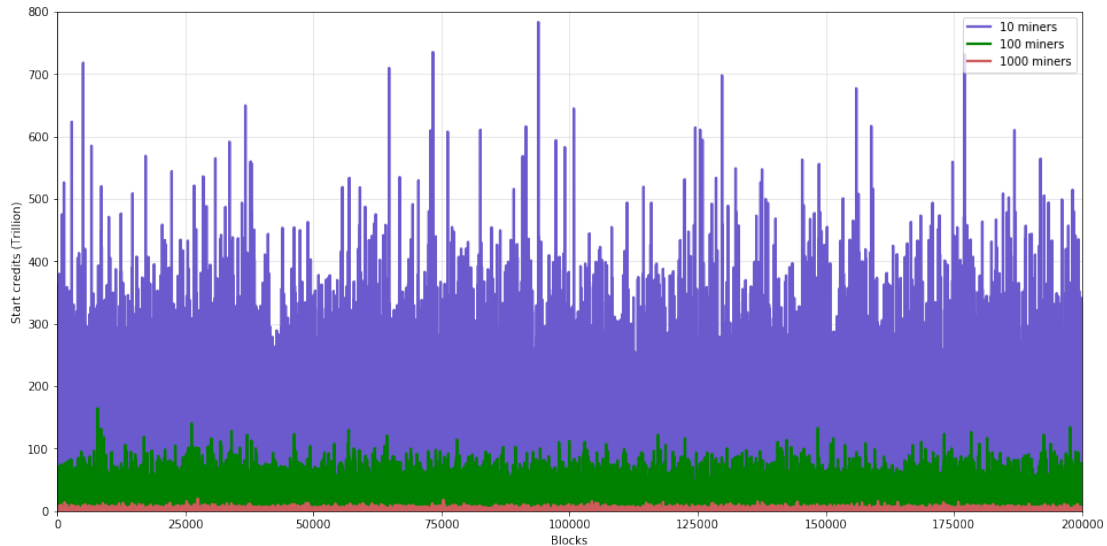


Figure 6.8: The number of credits the top miner was reset to for every mined block over 200 000 mined blocks in a 10, 100 and 1 000 miner pool.

The hash rate distribution of a pool may thus be very telling in the sense that it directly affects the credit differences between miners. Figure 6.9 shows for each of the three simulated pool sizes, the credit reset balance of a winning miner for each of the 200 000 mined blocks in a scenario in which no mining strategy was pursued⁹. For the 10 miner pool, the average reset balance was 45.52 trillion credits (TC) and hence significantly higher than for miners in the 100 and 1 000 miner pools. In the simulation of the 100 miner pool, the average reset balance for a miner was 9.83 TC, whereas for the 1 000 miner pool, a miner’s average reset balance was only 0.78 TC, less than 2% of the average reset balance of the 10 miner pool. The reset balance variance is also significantly higher in the 10 miner pool: $3.1241e+27$ compared to $1.3739e+26$ and $1.0804e+24$ in the 100 and 1 000 miner pools, respectively.

Even though the variance is high, the results show that miners who win a round in the 10 and 100 miner pools obtain substantially higher reset balances than in the 1 000 miner pool. The same data displayed in Figure 6.8 is presented by Figure 6.9, however, the reset balances have been sorted by size and are shown in descending order for each of the three simulated pools. Again, it can be seen that the reset balances were not only significantly higher in the 10 miner pool, but also the range of reset balances was substantially larger than for the 100 or 1 000 miner pool simulations.

The majority of the considered mining strategies focused on exploiting the non-uniform credit resetting mechanism of the queue-based scheme. Strategies like ExpVal2 specifically aimed at an attacking miner maximising his reset balances less the cost of giving up a specified number of rounds. The observed reset balances across all three pools may be a justification as to

⁹The y-axis label of Figure 6.9, *Start credits*, refers to the reset balance of a miner who wins a block

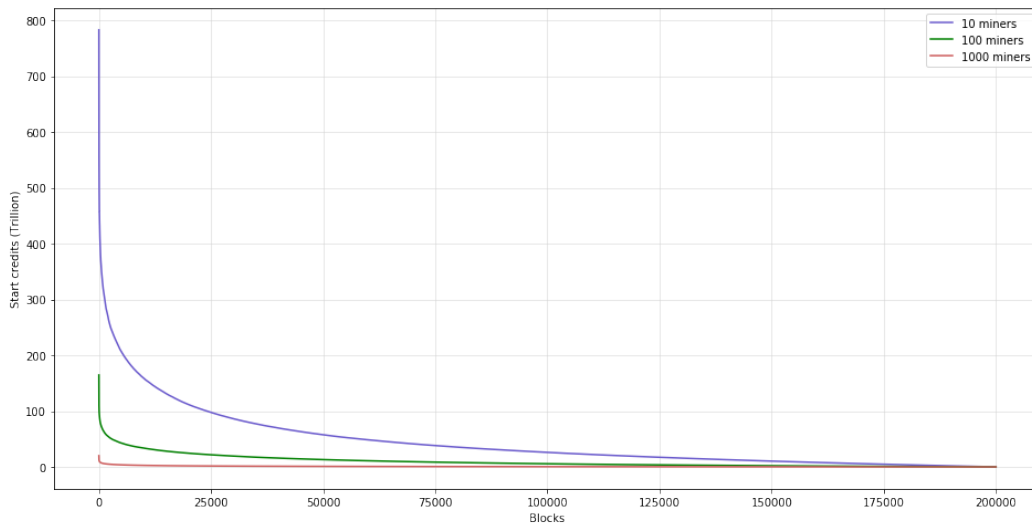


Figure 6.9: The credits a miner has been reset to for 200 000 mined blocks, sorted in descending order for pool sizes of 10, 100 and 1 000 miners.

why the ExpVal2 strategy’s performance varied so much across the three pools. High reset balances, as seen in the simulation of the 10 miner pool, would be a favourable prerequisite for employing a strategy which aims at maximising the former for a given miner. The considerably high variance of the reset balance in the 10 miner pool suggests that a miner could end up with a reset balance significantly larger than the average, by employing such a strategy. In fact, in the 10 miner pool, the highest obtained reset balance of a miner was almost 800TC, as shown in Figure 6.9. Such a high reset balance could result in a miner being close to the top of the queue again, as soon as his credits have been reset. When looking at the observed reset balances of the 1 000 miner pool, such a scenario seems rather unlikely given the low reset balance average. Additionally, even if a reset balance that would set a miner to the top of the queue was achieved, a great number, relative to the smaller pools, of other relatively large miners would potentially eat into this buffer. The low variance of reset balances in the 1 000 miner pool further suggests that the likelihood for an end of round credit difference between the top two miners being significantly greater than the average reset balance is rather unlikely. This is also suggested by the frequency of the underlying ExpVal2 condition being active, i.e. an expected reset balance being big enough for a miner to benefit from giving up one or more rounds, for each of the three simulated pools. In the 10 miner pool, for every 24th share submitted by the attacker the ExpVal2 condition was active compared to every 295th share in the 100 miner pool and every 1966th share in the 1 000 miner pool. This suggests that end of round credit differences between the top miners were favourable more often for an attacking miner in a small pool than in the larger pools. Hence a miner trying to maximise his reset balance could potentially benefit more from mining in a queue-based pool with a high average reset balance and a high variance, than mining in a queue-based pool with a low reset balance average and variance. It can thus be derived that a high variance and average reset balance may pose favourable conditions, whereas a low variance and average reset balance are adverse for mining strategies aiming to exploit the credit-resetting mechanism of queue-based mining pools.

A high average and variance of the reset balance may thus also explain the effectiveness of the pool hopping strategy in the 10 miner pool. The second 10 miner pool used for the

hopping strategy was also populated with miners that were generated by sampling for the hash rates. However, this second pool had an even higher average and variance of credit reset balances than the first 10 miner pool, namely 64.24 TC and $4.2566e+27$, respectively. Hence for the hopping scenario between the two constructed 10 miner pools, the attacking miner was essentially mining in two pools, which both posed favourable reset balances for miners, even if an employed mining strategy did not explicitly aim at exploiting the former. The extra blocks the attacker received in the 10 miner pool hopping scenario may thus be accredited to the high average and variance of reset balances in the second pool, as well as to the attacker for not giving up any rounds.

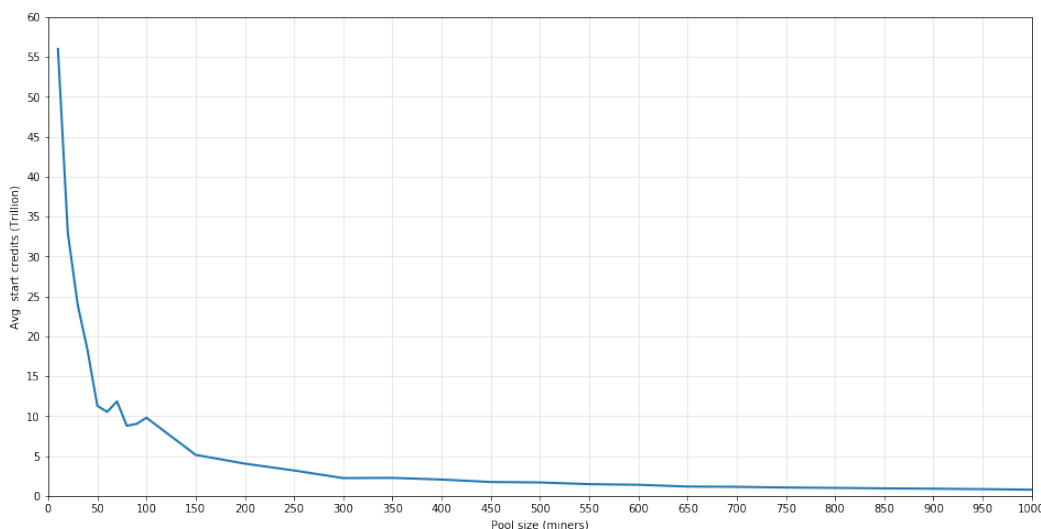


Figure 6.10: The average credits a top miner is reset to after 200 000 mined blocks for pools of various sizes of miners that have been generated from sampling from a log normal distribution of hash rates.

Given the potential relationship between the effectiveness of mining strategies and the reset balances of queue-based pools, the underlying determinants of these balances should be further examined. The presented findings suggest that the hash rate distribution of a pool, and hence the hash rate differences between the miners of a pool, directly affect the reset balances. In order to further examine this relationship, simulations of pools of various sizes have been run for 200 000 mined blocks and the reset balances have been recorded for each. The average reset balances for each of the simulated pool sizes are displayed by Figure 6.10.

Given that all miners in the simulated pools are created via the previously explained sampling method, as the number of miners in a pool (the sample size) increases, the hash rate distribution of the pool approaches a log normal distribution. Taking that into account, a general trend can be made out: as the pool size increases, the average reset balance decreases. The slight oscillations for when the pool size is between 50 and 100 miners are due to the relatively small sample sizes of miners and the risk of constructing pools with very different hash rate distributions. For example, a small pool in which the hash rates of miners lie closer together may experience lower reset balances than a small pool in which hash rates are further apart.

The reset balance variance for the same pool simulations as presented by Figure 6.10 is shown by Figure 6.11. Again, a trend can be seen: as the pool size increases, the reset balance variance decreases.

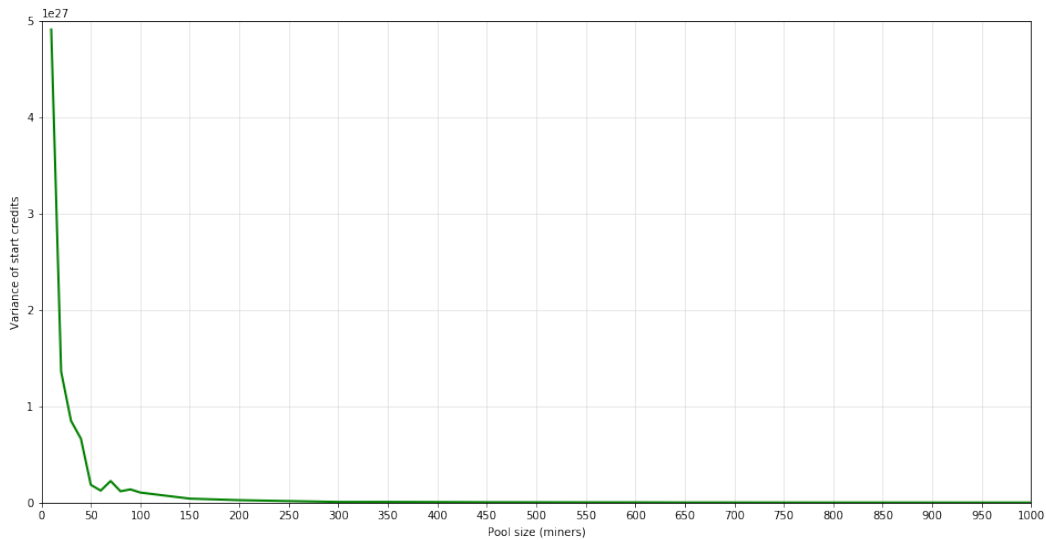


Figure 6.11: The variance of the credits a top miner is reset to for a simulation of 200 000 blocks for pools of various sizes.

The trends shown by Figures 6.10 and 6.11 are in line with the previously discussed findings for the 10, 100 and 1000 miner pools. As the pool size increases and the distribution of hash rates increasingly starts to resemble a log normal distribution, the average reset balance for a miner, as well as the reset balance variance decrease. This consequently affects the effectiveness of mining strategies that are aimed at exploiting the credit resetting mechanism of queue-based pools by maximising their expected reset balance over a given number of rounds. Under these conditions, less favourable queue constellations, which could be exploited by an attacking miner with an above average hash rate, arise.

These findings are based on a number of assumptions. It has been assumed that a queue-based mining pool's distribution of hash rates is log normal based on data obtained from Ethpool. However, as mining activity continues to increase and Ethpool experiences an increase in miners, it could well be that the hash rate distribution deviates from a log normal distribution. Furthermore, it has been assumed that a maximum of one miner can employ a mining strategy in a pool. In the real world of cryptocurrency queue-based mining pools this could very well differ as there is no limit for the number of miners that could engage in strategic mining. Lastly, it should be taken into account that all results were obtained from conducted event-based simulations. Even though the theoretical workings have been found to be correct, factors such as a miner's network connectivity, lags, or downtime of mining equipment have not been accounted for.

Chapter 7

Conclusion

7.1 Summary of Achievements

This dissertation addressed the workings of queue-based payout schemes implemented by cryptocurrency mining pools and examined the effectiveness of mining strategies aimed at exploiting the scheme's underlying credit-reset mechanism.

An event-based simulator was developed in order to model the behaviour of queue-based mining pools of various sizes over a specified number of mined blocks. Furthermore, the simulation framework allowed for proposing and simulating different mining strategies which could be employed by miners with above average hash rates. The effectiveness of such strategies was consequently examined for pools of different sizes. Noticeable performance differences of an attacking miner in a two miner pool and a multi-miner pool have highlighted the importance of well-defined conditions that determine the actions an attacking miner employs. It has thus been proposed that mining strategies which aim at exploiting the credit reset mechanism of the queue-based payout scheme should take into account the other miners' hash rates, the current queue constellation, as well as the work that has been performed by a pool during a round. Additionally, as the ExpVal2 strategy stated, the expected credit cost for an attacking miner giving up a round should be taken into account at all times.

Furthermore, it has been found that the effectiveness of such queue-based mining strategies, depends on the reset balances of a pool. Simulation results have shown that a high variance and average reset balance may pose favorable conditions for attacking miners, whereas a low variance and average reset balance are adverse for mining strategies aimed at exploiting the non-uniform reset balances. An underlying assumption to the simulations is rooted in the observed hash rate distribution of Ethpool resembling a log normal distribution. Under the assumption that large queue-based mining pools resemble a log normal distribution of hash rates, pools of different sizes have been constructed. Simulation results have led to the conclusion that as the hash rates of a pool approach a log normal distribution, the average reset balance and variance decrease. This directly negatively affects the effectiveness of mining strategies which focus on the non-uniform credit reset mechanism.

7.2 Applications

With the increase in cryptocurrency mining and the introduction of a queue-based payout scheme, understanding the benefits and potential risks of such a scheme is essential for participating miners. The queue-based mining pool scheme insights that stem from this dissertation could benefit the cryptocurrency mining community in several ways.

Miners Cryptocurrency miners with above average hash rates mining in a queue-based pool, i.e. Ethpool, can potentially increase their rewards by employing mining strategies under certain favourable conditions. Based on the findings presented by this dissertation, an attacking miner may be able to partially predetermine the effectiveness of strategies aimed at exploiting the non-uniform credit resetting mechanism by analysing a queue-based mining pool's hash rate distribution.

Academia Researchers with an interest in the workings of queue-based mining pools could benefit from using EthSim, the queue-based mining pool simulator which was created as part of this dissertation. The presented simulations have shown that EthSim can be used reliably to obtain data in order to evaluate the effectiveness of different mining strategies, as well as the performance of miners in a queue-based mining pool. Therefore EthSim has multiple applications for any researcher or individual with an interest in queue-based mining pools. The framework could be used for the simulation of newly proposed and implemented mining strategies, or for the further examination of the proposed strategies by constructing pools from the latest obtainable Ethpool miner data.

7.3 Future Work

In this section potential areas of research are discussed, which would further add to the findings presented throughout this dissertation.

7.3.1 Formal methods

It has been found that there is a clear trend that the end of round credit differences between the top two miners decreases for larger pools where the distribution of hash rates is log normal. However, rather than simulating various pool sizes, a mathematical model which would formally use a pool's hash rate distribution to determine the expected end of round credit difference would be very beneficial. Furthermore, a formal analytical model proving the workings of the queue-based payout scheme and underlying credit resetting mechanism would add further value to the cryptocurrency community, researcher, and possibly to the field of queuing theory.

7.3.2 An in-depth pool-hopping analysis

Theoretical pool hopping scenarios for queue-based mining pools have only been discussed briefly. Research focusing on the potential vulnerability of mining in multiple queue-based pools, specifically when a pool hopper takes into account the luck, hash rate distribution and queue constellation for every pool he mines in could turn out to be very interesting. Such work could also shed light on the potential effects of not setting the credits of a miner

who has not submitted shares for a longer period of time to zero. Currently, Ethpool does not reset the credits of a wallet. Understanding the broader implications of queue-based pool-hopping through a formal analysis could prove to be very useful for the cryptocurrency mining community, as well as for queue-based mining pool operators.

7.3.3 New mining strategies

This dissertation highlighted how an event-based simulator can be used in order to simulate the performance of mining pools for different mining behaviour. The notion of queue-based mining strategies was introduced and both, naive and expectation-based, strategies have been considered and simulated. However, the strategies examined throughout this dissertation are by no means all possible strategies a miner could pursue. In fact, a proper strategy tailored towards pool hopping using a condition that takes into account the states of multiple pools could be proposed.

Should additional queue-based mining pools emerge then given that research on queue-based mining strategies continues, cryptocurrency miners of all sizes could be in a position to determine which pool suits their needs and ambitions best. The future growth of cryptocurrency mining and the evolution of queue-based mining pools will certainly be an exciting field to watch.

Bibliography

- [1] Drand48. <http://man7.org/linux/man-pages/man3/drand48.3.html>. Accessed: 2017-09-04.
- [2] Etherscan. <https://etherscan.io/chart/difficulty>. Accessed: 2017-09-04.
- [3] Ethpool mining pool. <http://ethpool.org/>. Accessed: 2017-09-04.
- [4] Ethpool reward payout scheme. <http://ethpool.org/credits>. Accessed: 2017-09-04.
- [5] Lognormal distribution. http://en.cppreference.com/w/cpp/numeric/random/lognormal_distribution. Accessed: 2017-07-22.
- [6] A. M. Antonopolous. *Mastering Bitcoin: Unlocking Digital Currencies*. Sebastopol, CA, 1 edition, 2014.
- [7] V. Buterin. Ethereum: A next-generation smart contract and decentralized application platform. <https://github.com/ethereum/wiki/wiki/White-Paper>, 2014. Accessed: 2017-06-24.
- [8] E. community. Ethereum mining rewards. <https://github.com/ethereum/wiki/wiki/Mining#mining-rewards>. Accessed: 2017-06-26.
- [9] N. T. Courtois and L. Bahack. On subversive miner strategies and block withholding attack in bitcoin digital currency. *arXiv preprint arXiv:1402.1718*, 2014.
- [10] Crypto Coin News. Cryptocoinsnews. <https://www.cryptocoinsnews.com/altcoin/>. Accessed: 2017-09-07.
- [11] Ethereum community. Ethash. <https://github.com/ethereum/wiki/wiki/Ethash>. Accessed: 2017-06-18.
- [12] Ethereum community. Ethereum mining. <https://github.com/ethereum/wiki/wiki/Mining>. Accessed: 2017-09-04.
- [13] I. Eyal. The miner’s dilemma. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 89–103. IEEE, 2015.
- [14] B. Johnson, A. Laszka, J. Grossklags, M. Vasek, and T. Moore. Game-theoretic analysis of DDoS attacks against Bitcoin mining pools. In *International Conference on Financial Cryptography and Data Security*, pages 72–86. Springer, 2014.
- [15] Y. Lewenberg, Y. Bachrach, Y. Sompolinsky, A. Zohar, and J. S. Rosenschein. Bitcoin mining pools: A cooperative game theoretic analysis. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 919–927. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

- [16] L. Luu, R. Saha, I. Parameshwaran, P. Saxena, and A. Hobor. On power splitting games in distributed computation: The case of bitcoin pooled mining. In *Computer Security Foundations Symposium (CSF), 2015 IEEE 28th*, pages 397–411. IEEE, 2015.
- [17] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. <https://bitcoin.org/bitcoin.pdf>, Dec 2008. Accessed: 2017-06-18.
- [18] M. Rosenfeld. Analysis of bitcoin pooled mining reward systems. *arXiv preprint arXiv:1112.4980*, 2011.
- [19] O. Schrijvers, J. Bonneau, D. Boneh, and T. Roughgarden. Incentive compatibility of bitcoin mining pool reward functions. *Financial Cryptography and Data Security*, 2016.
- [20] A. Zamyatin, K. Wolter, S. Werner, C. Mulligan, P. Harrison, and W. Knottenbelt. Swimming with fishes and sharks: Beneath the surface of queue-based ethereum mining pools, 2017. viewed 10 July 2017.

Glossary

Altcoins Cryptocurrencies which can be seen as an alternative to Bitcoin, to the extent that each altcoin aims to improve some component of Bitcoin[10]. At the time of writing more than 1 100 altcoins existed.

Attacker A miner in a queue-based mining pool that pursues some behaviour or action with the intent of altering the queue constellation of miners in order to benefit from the non-uniform credit reset mechanism.

Block A data structure containing a list of transactions between nodes in the network. Blocks are the fundamental building units of the blockchain and consist of (amongst other fields) a block header, a time stamp, a nonce, the difficulty and the hash of header of the block it is being appended to.

Blockchain A distributed ledger in a peer-to-peer network, which is publicly accessible by all network participants (in the context of Bitcoin and Ethereum). The blockchain consists of blocks, where each block, or data structure, contains a list of confirmed transactions and is linked to the previous block in the chain via the hashed header of the former.

Condition and action scheme A mechanism for defining and implementing mining strategies in queue-based mining pools. A specified condition is checked each time an attacking miner submits a share. Should the condition be true, then the action which has been associated with the condition will determine the behaviour of the miner. Conditions could focus on current or expected queue-constellations, whereas actions relate to the way a share is handled by a miner.

Credits In a queue-based mining pool, a miner receives for every share he submits to the pool operator a number of credits equivalent to the share difficulty. In a queue-based reward scheme, the miner with the highest accumulated credit balance receives the entire reward (less pool fees) when a block is found by the pool.

Ethash The Proof-of-Work algorithm used in Ethereum which requires a miner to find some nonce input to the algorithm, such that the result is below a specified value (the PoW difficulty).

Hash A potential solution computed by a miner to the Proof-of-Work cryptographic puzzle, i.e. the PoW in Bitcoin.

Hash rate The computational power of an individual miner over a given period of time, generally seconds, expressed as either megahashes (MH), gigahashes (GH) or terahashes (TH).

Miner Nodes in the peer-to-peer network of a cryptocurrency such as Bitcoin or Ethereum, which try to solve the Proof-of-Work in order to be allowed to append the next block and hence receive the associated block reward.

Mining scenarios A queue-based mining pool in which no miner employs any mining strategies can be referred to as the normal scenario. Queue-based pools in which one or more miners do pursue mining strategies are alternative scenarios. For simulation purposes in this thesis, different scenarios were created and compared.

Mining strategy A miner pursuing an action when a specified condition is met in a queue-based mining pool, aiming to increase his reward payout.

Nakamoto consensus A common name for Bitcoin's decentralised consensus mechanism. Consensus within the network is needed in order to determine which miner node gets to append the next block to the blockchain, and hence alter the true state of the chain. The consensus mechanism predominantly consists of the Proof-of-Work, a cryptographic puzzle, which needs to be solved by some miner in the network.

Network difficulty A numerical value which must be greater than a potential solution to the Proof-of-Work in order for the solution to be valid.

Normal scenario A queue-based mining pool in which all miners aim to maximise their computational efforts and no miner tries to pursue any mining strategy.

Pool fee A small proportion of a block reward, collected by the pool operator to compensate his efforts. At the time of writing, the pool fee in Ethpool was 1%.

Pool-hopping A miner who switches between two or more pools in terms of the shares he submits to a pool, based on some favorable condition.

Proof-of-Work A computationally intensive puzzle requiring a miner to find some valid input, or nonce, to an algorithm (i.e. SHA-256 or Ethash) such that it is below a specified threshold, the network difficulty.

Queue-based attacks Actions employed by miners in a mining pool using a queue-based reward scheme in order to potentially exploit the underlying credit resetting mechanism.

Reset balance The number of credits a winning miner is reset to. A winning miner refers to a miner in a queue-based mining pool, who had accumulated the highest credit balance out of all miners in the pool when a block was mined by the pool.

Reward payout scheme A mechanism implemented by the mining pool that defines the distribution of rewards amongst the miners of the pool.

Rounds A round refers to the time period between one block found by a mining pool to the next one found by the same pool.

Share A solution to a Proof-of-Work (PoW) puzzle with a considerably lower difficulty than the network PoW, which has been set by the operator of a mining pool. This allows for a pool operator to track how much work each miner in the pool as performed over a given period of time.

Smart contracts In Ethereum, smart contracts can be described as programs that govern the transfer of Ether. These programs are written in solidity, a Turing-complete programming language called Solidity, which was created by the Ethereum Foundation.

Transaction fee The difference between the total inputs and outputs of all transactions in a block, which may be received by the miner who appends the block.

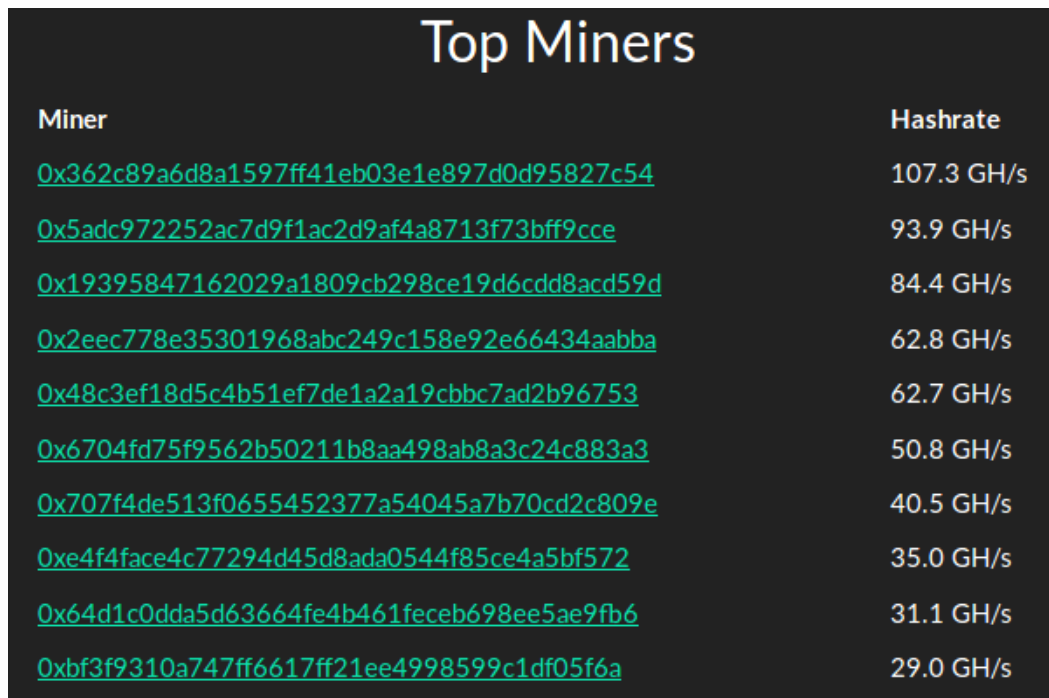
Transactions The transfer of some token, i.e. Bitcoin or Ether, between two wallet addresses. A transaction becomes confirmed once it is contained in a block, which has been appended to the longest blockchain.

Uncle A valid block found by a miner, which does not become the new head of the blockchain. This may occur when a competing miner also finds a block and benefits from his faster network connectivity.

Appendices

Appendix A

A.1 Ethpool



Miner	Hashrate
0x362c89a6d8a1597ff41eb03e1e897d0d95827c54	107.3 GH/s
0x5adc972252ac7d9f1ac2d9af4a8713f73bff9cce	93.9 GH/s
0x19395847162029a1809cb298ce19d6cdd8acd59d	84.4 GH/s
0x2eec778e35301968abc249c158e92e66434aabba	62.8 GH/s
0x48c3ef18d5c4b51ef7de1a2a19cbbc7ad2b96753	62.7 GH/s
0x6704fd75f9562b50211b8aa498ab8a3c24c883a3	50.8 GH/s
0x707f4de513f0655452377a54045a7b70cd2c809e	40.5 GH/s
0xe4f4face4c77294d45d8ada0544f85ce4a5bf572	35.0 GH/s
0x64d1c0dda5d63664fe4b461feceb698ee5ae9fb6	31.1 GH/s
0xbf3f9310a747ff6617ff21ee4998599c1df05f6a	29.0 GH/s

Figure A.1: The top ten Ethpool miners by hash rate for a given time on 06-09-2017.[?]

A.2 Hash rate distributions

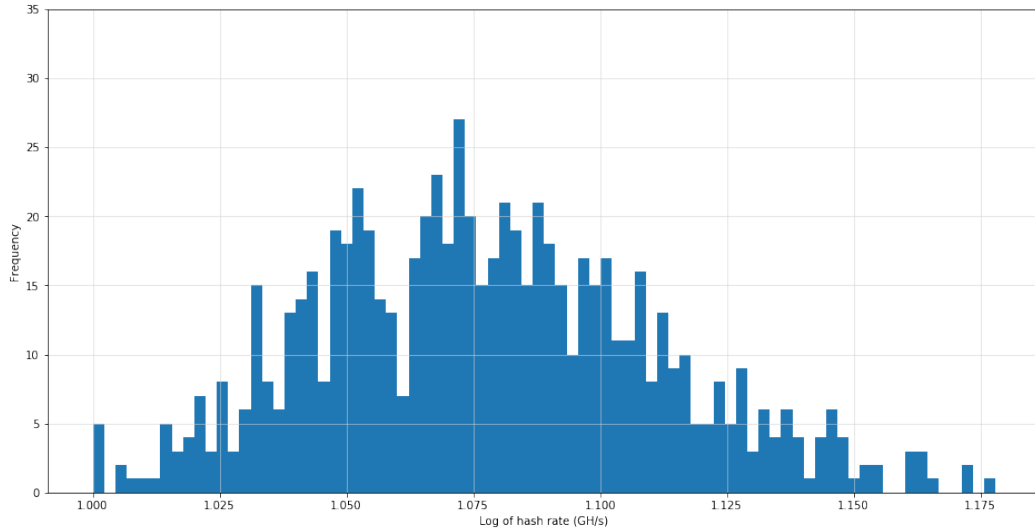


Figure A.2: The distribution of the logs of the hash rates for 729 miners in Ethpool resembling a normal distribution.

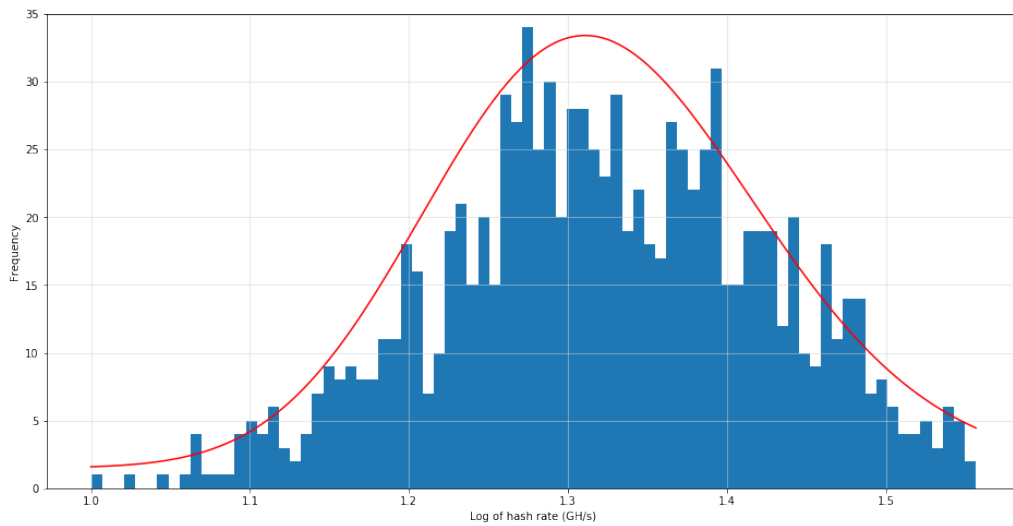


Figure A.3: The distribution of the logs of the hash rates for a constructed pool of 1000 miners, generated via sampling.

Appendix B

B.1 Design: EthSim

