PoolSim: A Discrete-Event Mining Pool Simulation Framework

Sam M. Werner[1,2] and Daniel Perez[1,2]

[1] Imperial College London, London, United Kingdom
[2] {sam.werner16,daniel.perez}@imperial.ac.uk

Abstract. In Proof-of-Work cryptocurrencies, fair reward distribution within mining pools has become a popular area of research. Aside from a theoretical grounding, mining pool reward scheme research has commonly involved discrete event simulations of deterministic miner behaviour under different reward schemes. However, until now researchers have been left with the tedious task of developing their own mining pool simulation software, a rather time-consuming and potentially extensive undertaking, as miner behaviour becomes more complex. We present *PoolSim*, an open-source and very extensible discrete event simulation framework for modelling different behaviours of miners under any reward distribution scheme. By utilising this framework for different hypothetical mining scenarios, we showcase that mining pool reward scheme analysis indeed remains an exciting area for future research.

Keywords: Mining pools · Discrete event simulation · Mining pool reward scheme analysis · Simulation tools.

1 Introduction

Ever since Satoshi Nakamoto first introduced the concept of Bitcoin [8] in 2008, the cryptocurrency landscape has turned into a \$134 billion[3] digital gold mine. This can be accredited to certain features Bitcoin first introduced, perhaps the most notable, a decentralised consensus mechanism. The consensus mechanisms is rooted in so-called *miners* trying to solve a computationally intensive cryptographic puzzle, referred to as the *Proof-of-Work* (PoW), where the difficulty is determined by the network. Miners are nodes which collect valid transactions while simultaneously trying to find a valid solution to the PoW problem. Solutions to the PoW are generated using modified versions of the block header to perform a preimage attack on the SHA-256 hash function in order to find a value which lies below some target threshold. The number of solution candidates, or hashes, a miner can compute per second denotes a miner's *hash rate*, typically expressed in megahashes[4] (MH/s) or gigahashes[5] per second (GH/s). The miner

[3] Total cryptocurrency market capitalisation. Source: https://coinmarketcap.com. Accessed: 2019-03-06
[4] 1 MH/s = 10^6 hashes per second
[5] 1 GH/s = 10^9 hashes per second

who finds a solution to the PoW is compensated for his invested computational effort in the form of newly minted units of the underlying cryptocurrency.

Cryptocurrency mining remains a fundamental structural component for the effective workings of the decentralised consensus mechanism in Proof-of-Work cryptocurrencies, such as Bitcoin and Ethereum [1]. However, rising difficulty levels of the cryptographic puzzles underpinning the mining process have posed severe constraints on the frequency of rewards paid to individuals trying to find PoW solutions. Hence, miners frequently pull their computational resources together to form mining pools, with the intention of reducing the high payout variance individual miners face. However, mining pools are faced with the non-trivial task of distributing rewards between pool participants in a fair manner, and in the past there has been much research[13, 10, 12, 5, 4, 7, 3] done on different mining pool reward schemes, as well as possible mining pool attack scenarios.

With the introduction of *PoolSim*, an open-source [9] discrete event simulation framework, researchers on mining pools and reward distribution schemes are no longer required to develop their own simulation software for modeling the behaviour of a miner under a particular mining pool reward scheme. Furthermore, by extending *PoolSim*, new mining pool reward schemes can be easily implemented and tested, and game-theoretic analyses conducted. To the best of our knowledge, there does not yet exist an actual simulation framework allowing users to fit each component to their needs by offering a high degree of extensibility, reliability, efficiency and good documentation. *PoolSim* is aimed at individuals interested in modeling different aspects of Proof-of-Work cryptocurrency mining pools with a focus on the reward distribution scheme employed.

The remainder of this paper is structured as follows. In Section 2 we explain cryptocurrency mining and outline the most common mining pool reward schemes. We subsequently examine some of the vulnerabilities identified in these reward schemes in Section 3. We explain the design of *PoolSim* in Section 4, prior to demonstrating its functionality by reproducing relevant existing academic work on mining pool reward schemes, as well as proposing potential avenues of future research in Section 6. We conclude in Section 7.

2 Preliminaries

In this section, we provide an overview of some of the most popular mining pool reward schemes, as examined in detail in [10] and [13].

2.1 Solo Mining

Miners mining independently are generally referred to as *solo miners*. Finding a block with some constant hash rate h follows a Poisson distribution with the rate parameter $\lambda = \frac{h}{D}$, where D is the network difficulty. Hence, a solo miner with a hash rate h who mines for time period t has an expected revenue of

$$E[R] = \frac{htB}{D} \tag{1}$$

where B is the block reward. A financial risk remains for a solo miner, rooted in the high payout variance the miner is faced with.

2.2 Mining Pools

In order to reduce payout variance, miners collude to form *mining pools*. These pools are typically run by a centralised pool operator who issues so-called *shares*, or PoW problems with a difficulty d, which is lower than the network difficulty. Each share has a probability of $\frac{d}{D}$ of being a PoW solution. Hence, the number of expected shares per block is qual to

$$E[S] = \frac{D}{d}. \tag{2}$$

The pool operator's task is to check whether a submitted share is a solution to the PoW puzzle and therefore whether a block has been mined. In order to compensate for his efforts, the pool operator retains a fixed proportion f of each block reward B mined by the pool. The remainder of the block reward $(1 - f)B$ is distributed among the pool participants according to some reward distribution scheme implemented by the pool operator.

2.3 Mining Pool Reward Schemes

In Bitcoin, a block reward amounts to BTC 12.5 per block, whereas in Ethereum the block reward equals ETH 2. In addition to receiving the transaction fees included in the block, in Ethereum, miners also receive rewards for mining or referencing uncle blocks. These are valid blocks that do not become the head of the longest chain. Ethereum rewards uncle blocks in order to incentivise miners to converge to a single chain, opposed to continue to mine on different branches in the event of a fork. The precise amount of the uncle reward is derived from the number of generations the uncle block lies away from the block which it is referenced by.

Fair distribution of block rewards within mining pools is not trivial. As multiple different reward distribution schemes have evolved over the years, we shall briefly discuss some of the most popular ones, as examined in more detail in [10] and [13].

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

Pay-Per-Share (PPS) In a *pay-per-share (PPS)* scheme, the pool operator pays a miner $(1-f)pB$, where p is the probability that the share is a PoW solution, i.e. $\frac{d}{D}$. By doing so, the operator fully absorbs the miner's payout variance. Hence, an operator of a PPS pool could make profits on short rounds, while being exposed to high losses on long rounds. Typically, in order to compensate for this risk, PPS operators charge higher fees compared to other reward schemes. For understanding optimal reserve balances in a PPS pool we point the reader to Rosenfeld [10], who formally examines this.

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

$$E[R_i] = (1-f)B\frac{s_i}{N} \cdot \frac{N}{D} \tag{3}$$

where s_i is the number of shares submitted by miner i during the last N shares [10].

Queue-based (QB) The *queue-based*[6] reward scheme was introduced by Ethpool, a small Ethereum mining pool. In a queue-based mining pool, miners receive an amount of *credits* equal to the difficulty of a share for each submitted share. When a block is mined by the pool, the full block reward[7] is allocated to the miner in the pool with the highest accumulated credit balance, namely the *top miner* in a priority queue. Subsequently, the top miner's credit balance is reset to the difference between his and the second highest credit balance in the pool. The reason for not resetting the credits of a top miner to zero is to provide an incentive for a miner to continue to perform work for the pool once he reaches the top position in the queue, opposed to switching to some other pool.

As first shown by [13], we provide a similar queue-based mining pool example in Table 1 showcasing that the credit resetting mechanism is non-uniform in the sense that the credits of top miners may be reset to differing balances. After a block has been mined by the pool (Table 1b), the top miner Bob has his credits reset to 10, the difference to Alice's credits. After each miner submits shares and receives credits, right before the next block is being mined (Table 1c), Alice is top of the queue. However, once the block has been mined (Table 1d), Alice is reset to a starting balance of 65 credits, a notably higher amount than Bob received. This suggests that Bob would indeed have been better off had he allowed Alice

[6] Ethpool refers to this as a predictable solo mining pool, however, we shall employ the term queue-based pool as introduced in [13]
[7] minus the pool operator fee

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

(a) Before Block i

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

(b) After Block i

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

(c) Before Block i+1

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

(d) After Block i+1

Table 1: The priority queue with miners of different sizes over a series of blocks.

to bypass him in the queue in order to receive a higher starting balance. We shall examine potential vulnerabilities rooted in this non-uniform credit reset mechanism in the next section.

3 Reward Scheme Vulnerabilities

Several mining pool reward scheme vulnerabilities have been identified in the past. In this section we outline what the main reward scheme-targeted attacks are and how these differ from one another.

3.1 Block Withholding

Research on possible attack scenarios between different mining pools has examined the effects of possible withholding attacks, whereby a mining pool mines in different mining pool and withholds blocks in order to cause direct harm to the pool [5, 4]. Furthermore, Schrijvers et al. [11] 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.

3.2 Pool-hopping

Apart from formally examining traditional reward schemes, Rosenfeld [10] also studies the effects of *pool-hopping*, whereby miners strategically decide in which pool to allocate their computational power in order to receive a reward higher than their fair share. Rosenfeld shows how unlike PPLNS, a proportional reward scheme is susceptible to pool-hopping attacks, as a miner is incentivised to leave the pool if a given round becomes too long due to the pool being unlucky. An examination of optimal pool-hopping behaviour and potential problems rooted in the prevention of such pool-hopping attacks has been conducted by [3, 2] and [6].

3.3 Queue-based Manipulation Attacks

Zamyatin et al. [13] examine potential vulnerabilities rooted in a queue-based mining pool and examine different strategies miners by the effects of different reward-increasing strategies a miner may pursue. These strategies are aimed at exploiting the non-uniform credit reset mechanism of the pool. As shown, in Section 2.3, a miner close to the top of the queue could be better off by allowing some other miner to surpass him in terms of total credits. By manipulating the queue order, a miner could aim to receive a higher credit starting balance once his credits have been reset. The authors identify three different actions a miner may take upon the submission of his share to manipulate the order of the queue, these being: share withholding, share donation and mining on a second wallet. For readability, we shall refer to the miner employing such behaviour as the *attacker*.

Share Withholding. The first action a miner may take to alter the queue order is to withhold computed shares from the pool operator. This has the effect that the attacker may be overtaken by some other miner with a higher credit balance in the queue. However, this approach suffers from inefficiency as the attacker performs work which is essentially lost. Furthermore, if the attacker is the top of the queue, there is no guarantee that the attacker will indeed be overtaken by some other miner in time before the round ends and his credits are reset.

Share Donation. In order to increase the likelihood of being bypassed by some miner, the attacker may donate his share to the address of the miner he intends to be overtaken by. The pool operator only receives the share and the address which should be credited for it and is therefore unable to make out donated shares. However, under this approach, the attacker suffers from missing out on performed work by giving shares away to other miners.

Second Wallet. In order to not miss out on performed work, an attacker may donate his shares to some other address, or wallet, of a set of addresses, which he controls in the same pool. Donating shares to a second address ensures that the attacker does not loose out on any performed work, while also being able to wait until he is overtaken by some other miner in the queue.

3.4 Queue-based Uncle Mining

A recent analysis of queue-based reward schemes has been conducted by Werner et al. [12], who use mining pool simulations to reconstruct real-world observations of a miner exploiting the uncle block reward distribution mechanism of a small Ethereum mining pool. Under the examined scheme, an uncle reward was allocated to some miner on a random basis. However, this supposedly random allocation did not take into account miners' hash rates and was thus susceptible to Sybil attacks, where miners would spread their hash rate across a large number of so-called *uncle traps*, low-hash rate miners which exist for the sole purpose of receiving an uncle reward. The authors show how miners in this particular queue-based pool are incentivised to deliberately withhold valid shares from the pool operator until a block has been mined by the network, in order to produce an uncle block and harvest the associated uncle reward.

4 PoolSim: Design and Implementation

Our system is implemented in C++ and is designed to be highly configurable and extensible. The proposed system is primarily composed of a shared library `libpoolsim` which provides the core functionality of the simulator and an executable `poolsim` which takes a configuration file as input, allowing for easily starting to run simulations. In this section, we describe the overall design of the system and provide some of the most relevant details about the implementation. We give a high-level overview of the design of the library in Figure 1.

PoolSim is a discrete-event simulator using a priority queue to store and execute scheduled share events. Share events represent shares generated by a miner, where the time intervals by which shares are submitted are constructed as a randomly generated exponentially distributed number with a rate parameter equal to $\lambda = \frac{h}{d}$, where h is the hash rate of a miner and d the share difficulty.

When a share is found, the miners is able to handle the share by using the provided share handler which can be configured or extended. The handler will usually submit the share to the pool it belongs to, but can choose any other behavior. When the mining pool receives a share, it delegates to the reward scheme, which can also be configured, to update the state and distribute rewards to miners. Once all this is done, the next scheduled miner processes the share it found, and this continues until the number of blocks in the simulation is reached. The results are finally serialized and include various metrics such as the number of blocks received per miner as well as the total work executed. Each reward scheme and miner behavior can define its own metrics which are serialized with the final results.

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

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

Extending PoolSim. One of the most important feature of PoolSim is its extensibility. There are two main parts which are designed to be very easily

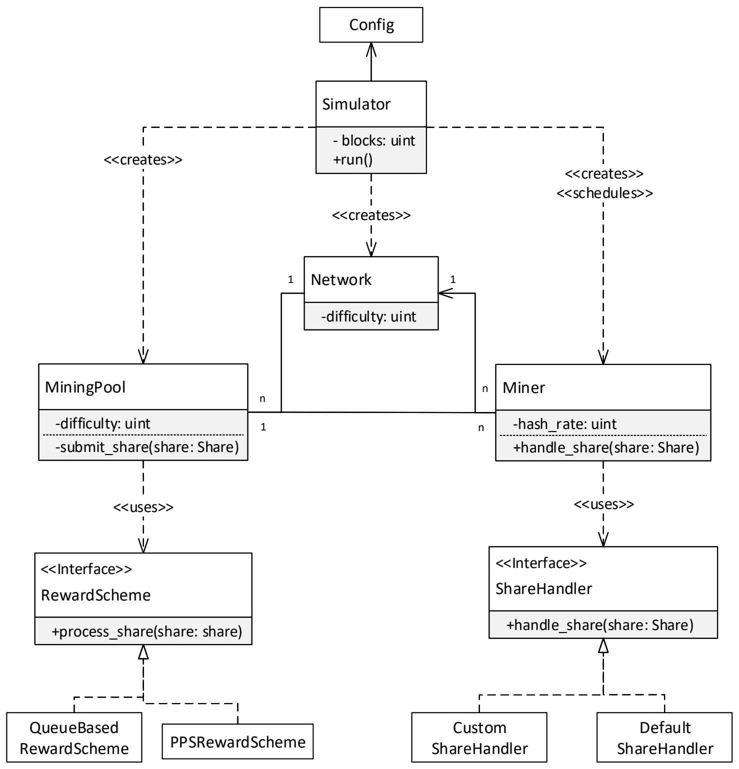

Fig. 1: Design overview of `libpoolsim`

extended: the mining pool reward scheme and the miner's share handler. In both cases, a base class is provided and custom behavior can be implemented by subclassing it. In Figure 2, we demonstrate the simplicity of creating new behaviors by presenting a share handler which withholds the share with the probability given in the configuration file.

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

5 Simulations

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

Listing 1: PoolSim configuration

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

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

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

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

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

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

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

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

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

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

5.1 Existing Research

We use *PoolSim* in an attempt to reproduce some of the key contributions of Zamyatin et al. [13][9]. In order to compare a miner's payouts of mining in a PPLNS scheme to mining in a queue-based scheme, [13] construct and evaluate the performance of a two-miner case, in which a small 1 GH/s miner and a large 10 GH/s miner mine under each scheme. Furthermore, the authors identify attack strategies (Section 3.3) specific to a queue-based pool, which may be employed by a large miner to potentially increase his payout in a two-miner scenario.

We reconstruct and simulate[10] the normal two-miner case, in which both miners mine honestly absent any attack strategies.

[9] We would like to remark that *PoolSim* has the capabilities of simulating each of the attack scenarios presented in Section 3, however, for brevity we shall only focus on the work by [13].

[10] It should be noted that we have made small modifications to the simulation setup compared to the original work for performance gains.

Two-miner Case: Normal. Zamyatin et al. [13] find that when comparing the performance between miners in the two-miner case under different reward schemes that the small miner is in fact performing less work per block than the large miner in a queue-based pool. The authors also show that the large miner would have been better off mining in a PPLNS pool (best option).

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

Table 2: Blocks mined and rewarded under solo mining compared to mining in a PPLNS and QB two-miner scenario.

In Table 2, we show our results of our reconstructed two-miner case. These are inline with the findings of [13], as it can be seen that the large miner performs a notable amount of more work than the small miner in the queue-based pool. This is reflected by the ratio of blocks received to blocks mined for the large miner in the queue-based pool, which is worse than had he mined in a PPLNS pool, or solo. The relatively high block ratio of the small miner for the queue-based pool is also inline with the findings by [13], indicating how the small miner benefits from the large miner absorbing the variance during lucky and unlucky streaks of the pool.

Two-miner Case: Attack. When looking at the attack strategies of share withholding, tactical donation of mining power and use of second wallets, the findings by Zamyatin et al. show that the optimal strategy for a large miner is the tactical donation of mining power in a two miner scenario.

We present the results for the two-miner attack scenarios in Table 3. As first discovered by [13], we also find that the tactical donation of mining power is the most successful strategy. By pursuing this strategy, the attacker is able to increase his block ratio to 1.029, compared to 0.993 in the normal scenario presented in Table 2, and thereby compensate for his initial loss.

It should be noted that our simulation configuration deviates from the work done by [13] when looking at the share withholding strategy. As a miner does not submit the share he withholds, the share is essentially lost, even if this share is a valid solution to the PoW. Therefore, the attacker will find less blocks during the same time period as for the other scenarios, in which no work is lost. In [13], the length of the share withholding is extended to equal the same duration as all the other scenarios. However, we believe that not submitting shares and thus mining fewer blocks poses the risk of being worse off compared to submitting all shares.

Attack strategy	Prop. of avg. credits lost	Miner	Avg. performed work per block	Blocks Rewarded	Blocks Mined	Blocks Ratio
Share withholding	0.188	Attacker	970 208	73 133	71 164	1.028
		Victim	1 295 488	7 005	8 974	0.781
Tactical donation of mining power	0.188	Attacker	972 887	91 192	88 610	1.029
		Victim	1 290 845	8 808	11 390	0.773
Using a second wallet	0.246	Attacker	1 008 641	90 215	90 906	0.992
		Victim	929 463	9 785	9 094	1.076

Table 3: Attack simulation results in a two-miner scenario.

Our findings for the effectiveness of the second wallet strategy deviate from the results presented in the original work, as the attacker in our simulation performed noticeably worse. In fact, the attacker would have been better off had he mined solo or pursued any of the other two attack strategies. The reason for this deviation from the expected results could be rooted in a simple check that needs to be performed by the attacker when deciding whether to donate his share to the second wallet. As stated earlier, a share is donated to the second wallet if the miner below the attacker in the queue has an accumulated credit balance of 90% or more of the attacker's balance. However, if this miner below the attacker in the queue is in fact the attacker's second wallet, then the attacker will get stuck in a loop of submitting shares to his two wallets on an alternating basis. Hence, the attacker needs to check whether the miner setting of the specified condition is indeed not his second wallet. We believe that this check was performed by [13], however, we did not have this check implemented when we obtained our simulation results and thus believe this to be the primary reason for the differing results regarding the second wallet strategy.

Simulation Performance In Table 4, we present the execution time of the different simulations performed for the two-miner scenario. It is worth noting that using our implementation, the ratio between the network difficulty and the pool difficulty influences greatly the speed of execution, as it changes the number of shares which must be generated before finding a block. As this ratio does not influence the behavior of the attacks we are checking for, we decide to keep it low in order to speed-up the simulations. In our simulations, this ratio is of 100, compared to about 20 000 for most real-world pools.

5.2 Normal Multi-miner and Attack Scenarios

In the previous subsection, we have examined how *PoolSim* can be used to reconstruct and examine the effectiveness of different queue-based attack scenar-

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

Table 4: Execution time of the different simulated two-miner scenarios.

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

Table 5: Multi-miner simulation scenarios.

ios. To receive further insights into the dynamics of queue-based mining pools we compare the performance of an attacker between mining in different pools containing 1 000 miners each. Table 5 shows the *PoolSim* execution time per simulated scenario.

When comparing the payouts of a miner under a PPLNS scheme to a QB scheme, we find that the attacker did in fact receive a slightly higher number of blocks in the QB pool (494) than in the PPLNS pool (488.31). We note that the attacker received in both pools a number of blocks higher than the number of blocks he actually mined. These types of analyses have indeed also already been conducted by [13]. However, we additionally simulate a scenario in which the attacker pursues the tactical donation of mining power strategy in a multi-miner pool. We selected the aforementioned strategy as this was the most effective one in the two-miner case. We find that the attacker received

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

Table 6: Execution time of the different simulated multi-miner scenarios.

less blocks (491) than had he mined honestly in the QB pool. The reason as to why the share donation strategy did not have any noticeable effect in this multi-miner pool is rooted in the existing credit differences between miners. The tactical donation of shares strategy requires certain credit differences to exist, as otherwise giving away shares does not pay off over time. In order to measure the extent of such differences in a given pool, we turn to the average proportion of credits lost per round. For a given round, this measure is expressed as the number of credits of the second miner as a proportion of the total sum of credits of the pool. Looking at this proportion, we find that in the scenario of share donation in the two-miner case, the average proportion of credits lost amounts to 0.188. Interestingly, for the scenario of share donation in a multi-miner case, we find that this figure amounts to 0.002, and is thus significantly lower. In fact, we suggest that the average credit loss is one of the main variable which can be taken advantage of by an attacker. For example, in a simulation with no attacker, the average credit lost is 0.245. Table 3 shows that successful attacks manage to reduce this average.

5.3 Queue-based Pool-hopping

This simulation demonstrates *PoolSim*'s capability to simulate condition-based pool-hopping scenarios. As briefly discussed in Section 3, several analyses on the effects of pool-hopping, as well as on which reward schemes are vulnerable to it, exist. However, we note that there have been no studies on the feasibility of pool-hopping between queue-based mining pools. Hence, we construct and examine a *proof-of-concept* pool-hopping scenario, where a miner submits shares to a pool on the basis of the luck of the pool. Pool luck for a given round r can be defined as

$$l_r = \frac{S_E}{S_A} \cdot 100 \tag{4}$$

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

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

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

6 Future Research Using PoolSim

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

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

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

7 Conclusion

Examining the exploitability of potential vulnerabilities embedded within different existing rewards schemes employed by mining pools has evolved into an interesting area of research within PoW cryptocurrencies. In this paper, we have introduced *PoolSim*, a simulation framework targeted for academics or anyone with an interest in studying and examining incentive and security mechanisms of mining pool reward schemes. Instead, *PoolSim* allows for a high degree of customisation, where new reward schemes could easily be implemented and tested, or the effectiveness of new attack strategies can be assessed accurately. *PoolSim*, finally allows researchers not having to deal with unnecessarily time-consuming and complex implementation tasks. We have provided an overview of the design of *PoolSim*, while also having demonstrated the framework's extensibility by reconstructing existing research focusing on discrete event simulations of mining pool reward schemes. Furthermore, we have used *PoolSim* to point out several potential areas of future work.

Acknowledgements. The authors would like to thank Paul Pritz for helpful feedback. This research is supported by funding from the Brevan Howard Centre for Financial Analysis.

References

1. Buterin, V.: Ethereum: A next-generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper (2014), https://github.com/ethereum/wiki/wiki/White-Paper, accessed: 2016-08-22
2. Chávez, J., Rodrigues, C.: Hopping among pools in the bitcoin mining network. The SIJ Transactions on Computer Networks & Communication Engineering (CNCE) **3**(2), 22–27 (2015)
3. Chávez, J.J.G., da Silva Rodrigues, C.K.: Automatic hopping among pools and distributed applications in the bitcoin network. In: Signal Processing, Images and Artificial Vision (STSIVA), 2016 XXI Symposium on. pp. 1–7. IEEE (2016)
4. Courtois, N.T., Bahack, L.: On subversive miner strategies and block withholding attack in bitcoin digital currency. arXiv preprint arXiv:1402.1718 (2014)
5. Eyal, I.: The miner's dilemma. In: Security and Privacy (SP), 2015 IEEE Symposium on. pp. 89–103. IEEE (2015)
6. Lewenberg, Y., Bachrach, Y., Sompolinsky, Y., Zohar, A., Rosenschein, J.S.: Bitcoin mining pools: A cooperative game theoretic analysis. In: Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems. pp. 919–927. International Foundation for Autonomous Agents and Multiagent Systems (2015)
7. Luu, L., Saha, R., Parameshwaran, I., Saxena, P., Hobor, A.: On power splitting games in distributed computation: The case of bitcoin pooled mining. In: Proc. 28th IEEE Computer Security Foundations Symposium (CSF 2015). pp. 397–411. IEEE, IEEE Computer Society, Verona, Italy (2015)
8. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf (Dec 2008), https://bitcoin.org/bitcoin.pdf, accessed: 2015-07-01
9. PoolSim: Poolsim. https://github.com/samwerner/PoolSim (2019), accessed: 02-03-2019
10. Rosenfeld, M.: Analysis of bitcoin pooled mining reward systems. arXiv preprint:1112.4980 (2011), https://arxiv.org/abs/1112.4980, accessed:2018-10-16
11. Schrijvers, O., Bonneau, J., Boneh, D., Roughgarden, T.: Incentive compatibility of bitcoin mining pool reward functions. Financial Cryptography and Data Security (2016)
12. Werner, S., Pritz, P., Zamyatin, A., Knottenbelt, W.: Uncle traps: Harvesting rewards in a queue-based ethereum mining pool. Cryptology ePrint Archive preprint: 2019/070 (2019), https://eprint.iacr.org/2019/070.pdf, accessed: 02-03-2019
13. Zamyatin, A., Wolter, K., Werner, S., Harrison, P.G., Mulligan, C.E., Knottenbelt, W.J.: Swimming with fishes and sharks: Beneath the surface of queue-based ethereum mining pools. In: 2017 IEEE 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS). pp. 99–109. IEEE, IEEE Computing Society, Banff, Canada (Sept 2017). https://doi.org/10.1109/MASCOTS.2017.22