Analysis of Selfish Mining Profitability via Monte Carlo Simulation

Maksim Khon
khonmaksim111@gmail.com

Abstract. The Bitcoin blockchain is a distributed ledger of transactions maintained by a network of nodes. The protocol assumes that honest nodes control a majority of the network's computing power. Conventional wisdom suggests that a minority group cannot earn revenue disproportionate to its hashing power, implying that the rational strategy is to remain honest. However, Eyal and Sirer (2013) challenged this view by introducing "Selfish Mining," a strategy that enables a minority pool to earn rewards exceeding its share of computing power. This paper replicates the original study using Monte Carlo simulations to verify the threshold at which this attack becomes profitable. The results confirm that a pool controlling more than 1/3 of the network hashrate can theoretically achieve higher returns than honest mining.

1. Introduction

Bitcoin is a decentralized cryptocurrency [1] that enables users to store and transfer value without reliance on third-party intermediaries. The network is built upon a blockchain—a distributed digital ledger composed of sequential blocks of transactions [2]. To append a block to the chain, a participant (miner) must solve a complex cryptographic puzzle. The miner who solves the puzzle first is rewarded with newly issued Bitcoins and fees from the transactions within that block. The probability of solving the puzzle is directly proportional to the miner's computational power.

The state of the network is determined by the longest chain of blocks, as it represents the greatest amount of accumulated computational power that has been allocated to construct it. Thus, the protocol requires the majority of miners to be honest. If a colluding group of miners or a single entity controls a larger share of the hashrate than honest miners, the network ceases to be decentralized, allowing that entity to determine the network's world state.

The frequency of rewards a miner earns is linearly proportional to their share of the network’s total mining power. Thus, miners often want to collude in mining pools to make their income more stable and predictable. The rewards in a mining pool are distributed among participants in proportion to each one’s contribution. Since a miner's total income is based solely on their mining power, they should earn the same average amount regardless of whether they are in a large or a small pool. Thus, conventional wisdom asserts that the Bitcoin protocol is incentive-compatible; that is, the best strategy is to follow the protocol, and a mining pool cannot have income disproportionate to its size.

However, Eyal and Sirer work [3] demonstrated that conventional wisdom is flawed, and a certain strategy, named “Selfish Mining”, employed by a minority pool can be used to obtain revenue exceeding its fair share. The standard protocol assumes an honest miner will broadcast its block immediately upon discovery; others, in turn, propagate it across the network, allowing every node to synchronize. However, a selfish miner keeps newly created blocks from broadcasting, secretly building on a private branch. Only when the public branch is about to catch up do they reveal this private chain. The network adopts the longer chain, causing honest miners to waste resources on blocks that are discarded.

This paper builds directly on the seminal work of Eyal and Sirer [3] and verifies their results using Monte Carlo methods.

2. Background

2.1. Selfish Mining

A miner is rewarded for a mined block only if the block ends up on the main (longest) chain, meaning other nodes build upon it. Consequently, it is in a miner’s best interest to publish a new block immediately to ensure it is accepted. It can then be concluded that a miner with hash power share x% is expected to receive x% of the total reward.

In contrast, selfish miners do not publish new blocks immediately. Instead, they withhold them to mine on a secret private branch. This strategy forces honest miners to waste resources on the public branch, mining blocks that are destined to be rejected. They achieve this by selectively publishing private blocks, causing the network to switch to their longer branch. This discards blocks mined by honest miners and allows the selfish pool achieve a reward share greater than their power share.

When the private branch is shorter than the public branch, selfish miners simply adopt the public branch, as the probability of them catching up is low.

When the selfish pool finds a block, it keeps mining on its private chain. There are two possible follow-ups: honest miners find a block, or the pool finds a second block.

If the honest miners find a new block, the selfish pool broadcasts its block immediately; a portion of the honest miners will recieve pool’s block first and consequently mine on it. This portion of honest miners is denoted by γ. Pool keeps mining on its head.

In the second scenario, where the pool finds a second block, it keeps mining on its head. For every block mined by honest miners pool reveals one private block. Once the pool’s lead reduces to one private block, it publishes its private chain. Since the private branch is one block longer than the public branch, the network adopts the private branch, and the process repeats.

Algorithm 1: Selfish-Mine

```
on Init
    public chain ← publicly known blocks
    private chain ← publicly known blocks
    $privateBranchLen \leftarrow 0$
    Mine at the head of the private chain.

on My pool found a block
    $\Delta_{prev} \leftarrow$ length(private chain) − length(public chain)
    append new block to private chain
    $privateBranchLen \leftarrow privateBranchLen + 1$
    if $\Delta_{prev} = 0$ and $privateBranchLen = 2$ then      (Was tie with branch of 1)
        publish all of the private chain                       (Pool wins due to the lead of 1)
        $privateBranchLen \leftarrow 0$
    Mine at the new head of the private chain.

on Others found a block
    $\Delta_{prev} \leftarrow$ length(private chain) − length(public chain)
    append new block to public chain
    if $\Delta_{prev} = 0$ then
        private chain ← public chain                           (they win)
        $privateBranchLen \leftarrow 0$
    else if $\Delta_{prev} = 1$ then
        publish last block of the private chain                (Now same length. Try our luck)
    else if $\Delta_{prev} = 2$ then
        publish all of the private chain                       (Pool wins due to the lead of 1)
        $privateBranchLen \leftarrow 0$
    else                                                       ($\Delta_{prev} > 2$)
        publish first unpublished block in private block.
    Mine at the head of the private chain.
```

Fig 1. Selfish Mining strategy proposed by Eyal and Sirer

2.2. Revenue

The pool receives a reward from a block only if that block is included in the main chain. The following are some of the possible scenarios analysed by Eyal and Sirer in their paper [3].

1. *A state with two branches of length 1.* If the pool finds a block, it publishes it immediately and earns revenue for 2 blocks. If honest miners find a block on top of the public branch, the pool gets nothing; others receive revenue of 2. If honest miners mine a block on top of the pool's head, the pool and the honest miners get revenue of 1 each.

2. *A state where the pool has a lead of 2.* If others find a block, the pool publishes all of its 2 private blocks; the pool receives revenue of 2.

3. *A state where the pool has a lead of more than 2.* If others find a block, the pool publishes one private block. Since the pool will eventually win (as seen in scenario 2), revenue of 1 goes to the pool.

3. Methodology

We analyze how relative pool revenue, defined as the ratio of selfish blocks that ended up on the chain to the total blocks on the chain, behaves with respect to the selfish pool's share of the network, denoted by α. In the case of a race between two branches, we split honest miners into two groups: one that mines on top of the pool's head (with ratio γ) and the other that mines on the public branch ($1 - \gamma$).

We define a state by its unique lead value, which is the difference between the private branch and the public branch, and use the Monte Carlo method to simulate transitions across those states. Transitions are accompanied by the distribution of revenue as prescribed in Section 2.2.

We build on the existing Selfish Mining model proposed by Eyal and Sirer (Fig. 1) and modify it to account for revenue distribution (A) (1).

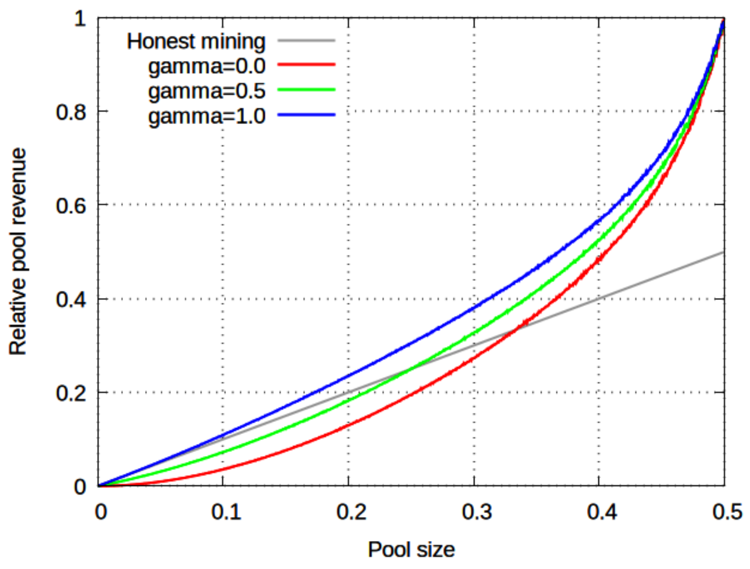

Fig 2. Relative pool revenue with respect to pool size (α) for different γ. Simulation of 1,000,000 block discoveries for $0 < \alpha < 0.5$

When the relative pool revenue exceeds the honest mining revenue, the pool earns a revenue share larger than its power share. The figure shows that for $\gamma = 0$ (i.e., when competing branches, all honest miners mine on the public branch), the selfish mining strategy is more profitable than honest mining for a pool size greater than 1/3 of the total network. On [1]the other extreme, where $\gamma = 1$ (meaning all honest miners receive the pool's block first), a selfish pool of any size will enjoy revenues higher than honest miners. For $\gamma = 1/2$, the selfish pool must hold a 25% share of the total network power to be more profitable.

4. Discussion

The simulation results indicate that 33% is the critical threshold required to successfully perform an attack on the Bitcoin blockchain, contrary to the 50% assumption previously held.

[1] The source code of implementation in C is available at github.com/khonmaksim/selfish_mining_sim

If honest miners choose between competing branches at random (i.e., $\gamma = ½$), this threshold is 25%. The results are consistent with the work of Eyal and Sirer.

Why were there no selfish attacks on Bitcoin? Operationally, selfish mining attacks create a measurable increase in orphaned blocks. Due to widespread awareness of this vulnerability, such attacks are likely to be detected before they become profitable [4]. Economically, the presence of an attack would erode trust in the system, causing the token price to crash and rendering Bitcoin profits worthless in fiat currency.

Limitations. This paper assumes a constant γ parameter, which is not true in the real network, where the network topology and propagation delays vary dynamically. Additionally, the study does not account for the duration of the attack, thereby excluding the potential effect of the difficulty adjustment algorithm [5].

5. Conlcusion

This study successfully replicated the discrete-event simulation proposed by Eyal and Sirer. The results confirm that a minority pool can, in theory, exceed its fair share of revenue. Specifically, the simulation validated the critical threshold of $\alpha = 1/3$ (or 33% hash power), above which a selfish miner earns excess profit regardless of network propagation γ. However, while the model shows the protocol's theoretical vulnerability, the possibility of such an attack remains uncertain.

References

1. S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System | Satoshi Nakamoto Institute," *nakamotoinstitute.org*, Oct. 31, 2008. https://nakamotoinstitute.org/library/bitcoin/
2. D. Yaga, P. Mell, N. Roby, and K. Scarfone, "Blockchain Technology Overview," *csrc.nist.gov*, Oct. 03, 2018. https://csrc.nist.gov/pubs/ir/8202/final
3. I. Eyal and Sirer, Emin Gun, "Majority is not Enough: Bitcoin Mining is Vulnerable," *arXiv.org*, 2013. https://arxiv.org/abs/1311.0243
4. K. Nayak, S. Kumar, A. H. Miller, and E. Shi, "Stubborn Mining: Generalizing Selfish Mining and Combining with an Eclipse Attack," *IEEE European Symposium on Security and Privacy*, Mar. 2016, doi: https://doi.org/10.1109/eurosp.2016.32.
5. M. Davidson and T. Diamond, "On the Profitability of Selfish Mining Against Multiple Difficulty Adjustment Algorithms," *Nist.gov*, Jan. 29, 2020. https://csrc.nist.gov/pubs/other/2020/01/29/selfish-mining/final (accessed Jan. 21, 2026).

A Monte Carlo Simulation

Algorithm 1 Selfish Mining: Monte Carlo Simulation of State Transitions

```
on Init
   pool lead ← 0
   privateBranchLen ← 0
   revenue pool ← 0
   revenue others ← 0

on runSim
   for number of iterations
          r = random float between 0 and 1
          if r < $\alpha$ then
                 poolFound()
          else
                 othersFound()

on poolFound
   privateBranchLen ← privateBranchLen + 1
   if pool lead = 0 and privateBranchLen = 2 then
          pool revenue ← pool revenue + 2  (was tie, pool wins due to lead of 1)
          privateBranchLen ← 0
   else
          pool lead ← pool lead + 1

on othersFound
   if pool lead = 0 and privateBranchLen = 0 then
          revenue others ←revenue others + 1      (was no private branch)
   else if pool lead = 0 and privateBranchLen ≠ 0 then
          compGammaParam()           (2 competing branches; determine which branch others build on)
          privateBranchLen ← 0
   else if pool lead = 2 then
          revenue pool ←revenue pool + 2   (publish all 2 private blocks)
          pool lead ← 0
          privateBranchLen ← 0
   else if pool lead > 2 then
          revenue pool ← revenue pool + 1  (publish one private block)
          pool lead ← pool lead - 1

on compGammaParam
   r = random float between 0 and 1
   if r < $\gamma$ then                (honest miners build on pool’s branch)
          revenue others ← revenue others + 1
          revenue pool ← revenue pool + 1
   else                        (honest miners build on public branch)
          revenue others ← revenue others + 2
```