Blockchain Consensus: Secure and Fast Transactions

Dae Kwang Lee

March 20, 2018

Abstract

In Bitcoin, transactions are stored in a timestamped object, or a block. Ideally, the public ledger forms a singly linked chain of blocks, or a single blockchain, and each user maintains a uniform state of the blockchain copy. When the blockchain has multiple branches, or forks, the longest branch is the only valid chain, according to the current protocol, the Nakamoto Consensus. GHOST is an alternative protocol that instead selects the branch with the most blocks appended to it. GHOST is much faster than Nakamoto Consensus in transaction processing but produces more forks. Neither protocol protects the system when selfish miners collude to keep mining on a shorter branch and make it the longest or the heaviest to revert valid blocks. What we need is a new policy that is as fast as GHOST but is as secure as the Nakamoto Consensus.

In this paper, we extend GHOST and propose an alternative policy, HIRES, which resolves forks by choosing the subtree containing the highest transaction fees, or residuals. We hypothesize HIRES is as fast as GHOST and as secure as the Nakamoto Consensus, as miners choose transactions with high residuals to maximize their incentives and resilience to reversion. We test the three protocols with extreme and typical block parameters in Bitcoin network simulator by Arthur Gervais to compare their efficiency and their security. In the experiment, we conclude that HIRES and GHOST are approximately 30 times faster than the Nakamoto Consensus, but the latter is the most resilient as it incentivizes the selfish miners the least.

Contents

1 Introduction

Bitcoin is the first successful decentralized electronic cash payment system, developed by Satoshi Nakamoto in 2008 and deployed in January 2009 [14]. Bitcoin has seen rapid growth over the past few years, both in value and the number of transactions. As of today, a unit of Bitcoin is worth more than 9000 USD, and more than 300,000 transactions are being processed daily [3]. Estimated transaction volume exceeds 1B USD, and more than 140 Bitcoin exchanges provide exchange services with alternative cryptocurrencies, digital assets, and real-world currencies [4]. The press predicts a continual rise of digital currency as more people distrust the stability of national currencies and government policies [13].

Under the meteoric rise of Bitcoin, decentralization is the core property; unlike a conventional e-cash scheme that requires a trusted third party to prevent double-spending, an act of spending the same money twice, and mediate disputes, Bitcoin relies on a peer-to-peer network where each user stores a common public ledger. Specifically, if a user Alice pays a user Bob, the corresponding transaction is broadcast to every user in the system. Each user, or *node*, decides whether the transaction satisfies a set of validation rules [15]. If the transaction is confirmed valid, the nodes place it in a queue, where all valid transactions are waiting to be added to the public ledger; invalid transactions are rejected.

Bitcoin achieves decentralization with its incentive mechanism. It rewards the nodes, called *miners*, who collect the pending transactions and put them in an object called *block*. Miners solve a complex computational puzzle, or *proof of work*, to make a valid block and connect it to a previously created block, or its *parent*. Miners usually share their computation power in a *mining pool* and jointly work to generate a block faster. Once they find the solution, the block is appended to its parent in the public ledger, where all the generated blocks are chained in chronological order since the very first block (*the genesis block*). Sometimes multiple blocks are made concurrently, and thus the public ledger is not necessarily a single line of blocks, called a *blockchain*, but forms a *block tree* with several branches.

Bitcoin nodes agree on a blockchain policy called *Nakamoto Consensus*, named after the creator of Bitcoin; the longest branch is the only valid transaction history. Thus, miners who add blocks to the longest chain are rewarded with newly minted bitcoins and the transaction fees from the blocks. Blocks in shorter branches, or *stale blocks*, are ignored. An example diagram is provided to more aptly describe a single

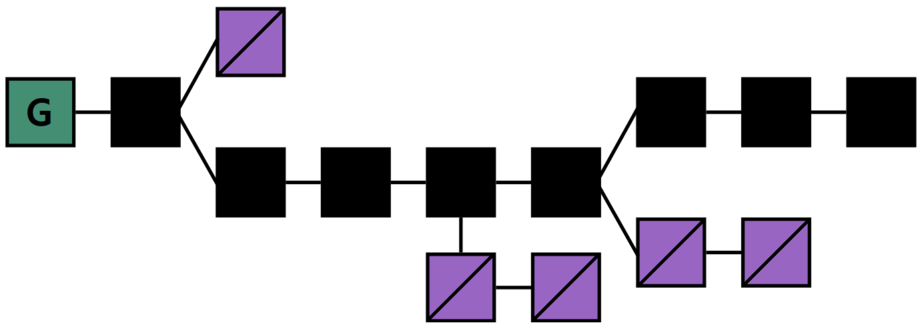

Figure 1: A block tree generated from the genesis (green) block.

blockchain. Figure 1 describes a block tree where the chain of filled blocks is the only valid chain [11]. The state of having multiple branches is said to have *blockchain forking*.

1.1 Blockchain Forking

Forking occurs when nodes are accepting different blocks based on their local copies of the ledger. The inconsistency is closely associated with nodes' activities over Bitcoin network. If multiple blocks of a single parent, or its *children* are generated, some nodes receive one of them first and discard the other while others do the opposite; miners do not agree on a single new parent for the next block to be mined, forming multiple branches with longer forks. In Figure 1, some nodes accept the second filled block and discard the first slashed block while others do the opposite; the temporary inconsistency is resolved by mining on the filled block; the fourth filled block and the second slashed block are in a fork, and it becomes longer as the next set of blocks are appended to different branches; miners agree on appending to the fifth filled block, and the nodes accept different blocks until the last filled block is mined.

We provide another example to demonstrate how the network nodes accept different blocks. Figure 2 describes a peer-to-peer network that emulates Bitcoin network [6]. A, B, C, and D refer to Alice, Bob, Charlie, and David, and they represent the nodes we mention in the diagram. Other nodes are depicted as filled circles, and an edge connecting two nodes signifies communication between them. Alice and Charlie

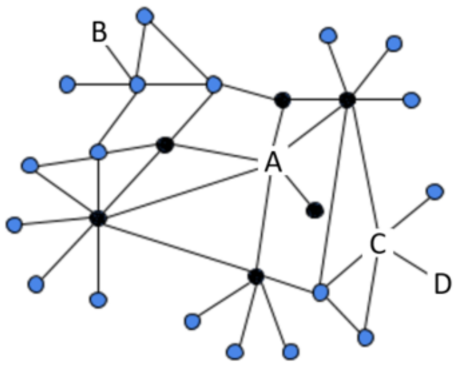

Figure 2: A peer-to-peer network.

are miners from different mining pools, and each has a newly generated block to broadcast. Bob and David are the most distant nodes waiting for the block.

Alice and Charlie have two valid blocks, and each contains the information of the miner, the unique ID of its parent, and a collection of transactions. Alice and Charlie send their blocks to their closest nodes, or their *peers*, via the edges that connect them. David receives Charlie's block sooner than Alice's block, so he accepts Charlie's block and discards Alice's block. Bob receives from Alice sooner than Charlie because Bob is closer to Alice. Likewise, Bob accepts Alice's Block and ignores Charlie's block. Now David and Bob have different local copies of the blockchain. If Charlie and Alice do not agree on selecting a new parent and append to their own previous blocks, the fork becomes longer as Bob and David validate the new blocks against their public ledger.

In reality, the Bitcoin network is much bigger and tangled; many nodes may have different local replicas of the blockchain, and a single forking may develop into multiple branches. Blockchain forking is a serious security threat as self-interested miners collude and beat the longest chain to revert valid transactions. In Figure 1, the attackers can add two consecutive blocks to the last slashed block and make the chain the longest, reverting valid transactions from the last filled block; the filled block becomes a stale block. If the attackers have enough computational power over the network, they can start catching up from few blocks behind and supersede the main chain, making work of honest miners wasted and receiving incentives dishonestly.

1.2 Nakamoto Consensus vs. GHOST

In practice, Nakamoto Consensus naively resolves forks by adjusting the hardness of proof of work; miners typically generate a 1MB block every 10 minutes. Block generation time is much slower than propagation time, so the nodes are not likely to receive two consecutive blocks, and the probability of forking decreases. In theory, the system can endure attacks based on forking, such as the attackers colluding to mine from few blocks behind to make their branch the longest and revert transactions in the main chain [14], or *selfish mining*.

In 2013, Yonatan Sompolinsky and Aviv Zohar proposed an alternative consensus protocol called the *GHOST*, which resolves forks by selecting the heaviest leading block at each fork rather than the longest chain [17]. In other words, a block that has more blocks appended becomes the block in the main chain. The authors hypothesize stale blocks can be included in the main chain to accelerate its growth. They propose significantly decreasing the size of a block and the block creation time by making proof of work easier to accelerate transaction processing: make a 320KB block every 1 second.

Yet, the policy based on GHOST generates blocks too fast that its forking rate is much higher than that of the current Bitcoin policy. For example, in Ethereum, another distributed platform based on blockchain technology that modifies GHOST to make a 1.5KB block every 10 to 20 seconds, 6.8% of blocks become stale blocks among 4000 nodes, whereas 0.41% exists in Bitcoin among 6000 nodes [10]. What we need is a policy that generates blocks as fast as GHOST and is resilient to attacks based on forking.

1.3 Research Question

Our research question delves into the following problem: Is there a policy that is as fast as the policy based on GHOST and as secure as the current Bitcoin policy based on Nakamoto Consensus?

1.4 Our Contribution

We use Arthur Gervais's Bitcoin network simulator [1] to compare two policies based on GHOST and Nakamoto Consensus. We conduct experiments under honest mining and selfish mining, each with extreme and typical block parameters, where the total statistics of mining activity and information propagation are returned for each simulation. For honest mining tests, we focus on the number of total blocks in the main chain, the number of stale blocks, and the average block propagation to compare the efficiency of each policy on the growth of the main chain. For selfish mining tests, we focus on the number of selfishly mined blocks and the selfish miner's contribution to the growth of the main chain to compare the security of each policy.

Our main contribution is to extend GHOST and to propose a new blockchain policy based on our analysis of the preliminary result. We propose HIRES that selects a block leading to the highest total value of transaction fees at each fork to make transaction subversion more difficult. Lastly, we evaluate three policies using the same approach and discover the impact of the forking rate on the efficiency and the security of Bitcoin blockchain.

2 Organization

The paper is organized as follows. In Section 3, we elaborate technical details of Bitcoin and introduce two Bitcoin protocols, Nakamoto Consensus and GHOST. In Section 4, we overview the related work on the different analysis of security threat on Bitcoin. In Section 5, we introduce Arthur Gervais's Bitcoin network simulator and the preliminary simulation result of Nakamoto Consensus and GHOST to discuss our method for the new protocol, HIRES. In Section 6, we evaluate the simulation results of Nakamoto Consensus, GHOST, and HIRES. In Section 7, we draw our conclusion based on the result. Section 8 discusses future work, and Section 9 is the acknowledgment.

3 The Bitcoin Protocol

This section explains technical details of the Bitcoin protocol to offer a better understanding of the research topic and our work. Section 3.1 elaborates how Bitcoin transactions are processed and validated. Section 3.2 describes the implementation of blocks and mining process. In Section 3.3, we focus on security threats and attack triggered by blockchain forking. Section 3.4 introduces Nakamoto Consensus, and Section 3.5 describes an alternative protocol, GHOST. Finally, in Section 3.5, we compare Nakamoto Consensus and GHOST in terms of selecting the next block to be included in the main chain.

3.1 Transactions

We give an example addressing Alice, Bob, and Charlie to describe a public key and a private key in Bitcoin. We assume Alice and Bob are regular nodes, and Charlie is an attacker. Each node has a private key, a 256-bit number, and a public key, a 512-bit number[1]. Alice stores her private key in a secure place and keeps it secret. Her private key tells that she is a valid user to spend her coin [2]. If Charlie steals her private key, then he has a permission to spend her coin. However, Charlie gains nothing by stealing Bob's public key as it signifies Bob's Bitcoin address.

We address Alice, Bob, and David to explain the input and the output of a Bitcoin transaction. Alice has 4 BTC[2] and wants to send all to Bob. Unless Alice is Satoshi Nakamoto, the creator of Bitcoin who is unidentified, Alice must have received 4 BTC or more from some node, David, during her previous transactions. When Bob receives the money from Alice, then Bob's money is actually from Alice who has made transactions with David to receive the money. Thus, the output of the previous transaction becomes the input of current transaction, and the output of current transaction is an instruction for sending money to the designated recipient.

We use Alice as a sender, Bob as a receiver, and Charlie as an attacker to describe how to make a valid transaction. Alice uses her private key to digitally sign a hash of the previous transaction and Bob's public

[1]Compressed public key is 257 bits, but this paper rather focuses on the use of public and private keys.
[2]a unit of Bitcoin

key [14]. Bob proves the ownership with his public key; If Charlie modifies the transaction content, then Bob knows because he cannot verify the digital signature of Alice with his public key. In reality, Alice has to leave some portion of her money as transaction fees, or *residuals*, when making the transaction to Bob, and Bob receives less than 4 BTC. The output of a transaction cannot exceed the input.

We use Alice as a sender to describe how transactions are validated. Alice can make multiple transactions at a time. Most importantly, she must send the rest of the output back to herself. If she decides to send only 3 BTC out of 4 BTC, then she has to send the rest back to herself. Once the transaction is validated, it is broadcast to the network nodes, just as in Figure 2. Each node validates the transaction with a set of validation rules; each transaction is a script that executes functions that check whether the sender uses a valid private signature to refer to the previous transaction and the public key of a recipient; it checks whether transaction fees are included; the last function checks whether the public key is the valid address of the recipient. Lastly, the network nodes check their copies of blockchain to prevent accepting two concurrent transactions using the same coin. After the transaction passes the rules, it is pushed to a memory pool, where all unconfirmed transactions are randomly organized to be processed. Finally, when the nodes confirm that the transaction is in the pool, they no longer broadcast it [15].

3.2 Blocks and Mining

A block, in the context of Bitcoin, is a timestamped file that permanently records transaction data pertaining to the network nodes [2]. To generate a valid block, miners use Bitcoin mining software to concatenate an arbitrary one-off number called nonce, the hash of its parent, and the list of transactions included in the block and calculate the hash of the entire thing such that the output falls in a target value, a 256 bit number starting with a certain number of zeros; the hash of a newly generated block must have enough leading 0's such that $\leq$ target. For instance, if the target value has 10 leading 0's with the remainder of random values, any block hash less than or equal to the target is considered valid.

In practice, Bitcoin adjusts the hash target every 2016 blocks [2] to maintain the average mining time of 10 minutes. The hardness of proof of work, or its difficulty, is the ratio of the maximum target, having

32 leading 0's and the remainder of 1's, divided by the current target. As of today, the entire computation power is approximately 9.584 ExaHash or $9.584{*}10^9$ GH/s, and the current difficulty is $3.3{*}10^{12}$ [3]. In other words, miners have to increment the nonce and calculate SHA-256, a hashing algorithm, about $3.3 * 10^{12}$ times as many hashes to find the solution. The major mining pools generate blocks even faster, which implies the amount and the value of computer resources to participate in mining are tremendous [12].

Once they find the valid nonce value, the block is generated and broadcast to the public. To avoid transmitting blocks to the nodes that have already received the data from other nodes, an *inv* message is sent to announce the availability of the transactions and blocks that have been validated. When a node receives an *inv* message from its peer or another node, it requests the transmission of the corresponding data by sending a *getdata* message. Then the announcer of the inv message sends the data. During information propagation, the propagation delay occurs because the size of the block message is much larger than that of *inv* and *getdata* messages. The delay costs for small blocks are greater because even the small message is announced via an *inv* message and retrieved via a *getdata* message [8].

When a node receives the block, he tests its validity. A valid block must have its hash starting with enough 0's that falls in a target value; the block must only have unconfirmed transactions, and having identical transactions with any previous blocks in the block tree disqualifies it being permanently added to blockchain; it must have the correct hash of its parent that is being appended to, or otherwise it becomes an *orphan block*, a block that has no parent. An orphan block may become the genesis block to form another chain, but the likelihood of it superseding the main chain is very low as miners keep adding to the main chain to receive the block reward. After validation, each node adds the block to her copy of blockchain and repeats the announcement procedure until the most distant nodes receive the block.

The Bitcoin system rewards the miners by minting new Bitcoins whenever they find the solution to proof of work. The system controls the circulation of coins by making the mining process more difficult as the rate of block creation increases and the average mining time decreases [7]. By regulating the difficulty of the mining process, the rate of block creation approaches its standard rate, and the average mining time increases. As a whole, a new block is generated every 10 minutes, and minting will reach the capped total of 21 million BTC by the year 2140.

3.3 Blockchain Forking and Attack

Ideally, the blockchain should resemble a singly linked list where an incoming block points to its previous block or its parent. However, many mining pools attempt to generate blocks faster than their counterparts, and sometimes two or more blocks point to the same parent. The network nodes receive different blocks at a different time and have a different state of blockchain copies. In extreme case, the inconsistency results in multiple branches and forms a tree of blocks, degrading Bitcoin's credibility [17].

In practice, the network nodes select whichever they receive first. Bitcoin assures that the average time it takes to broadcast a newly generated block to the network nodes is negligible to the expected time between consecutive block generation [14], and any kind of temporary inconsistency eventually synchronizes to share a uniform copy of blockchain [8]. Nonetheless, scholars argue whether blocks would need to become larger as Bitcoin would process more transactions, and longer propagation delay would increase the likelihood of blockchain forking [17].

The less problematic type of attack based on forks is *double spending*. We recall Alice, Bob, and Charlie in Section 2.1. Suppose Charlie sends Alice 1 BTC, but later he sends the same coin to Bob. Some miners create a block includes C-A[3] transaction, while others create a block includes C-B[4] transaction. The common heuristic to minimize the vulnerability of the attack is to wait for six blocks to be added after receiving or making a payment to verify their transaction has been accepted. Alice and Bob wait for six blocks to be appended on top of their transactions; one of them receives 1 BTC from Charlie as the network nodes notice the inconsistency among their copies and agree on one transaction.

The most renown attack based on forks is *51% attack*; malicious miners occupy more than half of the computation power and jointly produce a block to dishonestly make a profit. Unlike honest miners who broadcast blocks upon creation, the attackers select a block in the main chain as a starting block and mine in their private chain. They release the private chain when it becomes longer than the honest chain appended to the attack source. The type of mining is called *selfish mining*, and if $>$ 50%[5] of the mining power is engaged in selfish mining, reverting valid blocks appended on the source block becomes possible. Thus,

[3]Charlie to Alice
[4]Charlie to Bob
[5]Ittay Eyal et al. proposes 33%, but we use 50% in this paper.

the common heuristic of waiting for six more blocks does not protect the system. To that extent, Satoshi Nakamoto assures the success rate to reverse a chain of block drops exponentially as the number of blocks appended to the target block increases [14]. However, blocks in forks, or stale blocks, may distribute honest computation power as honest miners add on different branches in the main chain while the attackers only work on adding to their private chain.

3.4 Nakamoto Consensus

Satoshi Nakamoto, the creator of Bitcoin, proposes Nakamoto Consensus [14] to resolve forking and thwart reversion of valid transactions. Nakamoto Consensus handles forks by selecting whichever is the longest as the main chain; shorter branches exist in the ledger, but miners choose the longest chain for mining to get incentives. When two branches are the same length, whichever has the next block appended becomes longer, and it becomes the main chain. The consensus reflects Bitcoin is secure as long as the attacker nodes do not exceed the half of the mining population.

In the Bitcoin white paper, Satoshi Nakamoto makes a scenario for selfish mining where the sender is an attacker who attempts to revert the transaction with an honest node. The author uses Poisson distribution to retrieve the probability of the selfish miner to start mining from n blocks behind and making his private chain longer than the existing main chain when the selfish miner occupies $< 50\%$ of the entire computational power over the network. The author concludes the value drops exponentially as n increases; when the selfish miner has 10% of the computational power, the probability of catching up after 5 blocks added to the target block is $\leq 0.1\%$; when the computational power is 45%, the probability is $\leq 0.1\%$ for n = 340.

3.5 GHOST

Yonatan Sompolinsky and Aviv Zohar [17] predict the rapid adoption of the digital currency and the necessity of accelerating Bitcoin's transaction processing. They point out two design flaws in Bitcoin blockchain policy, the propagation delay of blocks and the waiting time for transaction confirmation, are restricting the rate of transaction processing significantly lower than the network bandwidth for streaming transactions. While Nakamoto Consensus ensures its secureness with its heuristic of waiting for six blocks to be added in the main chain, the authors argue its confirmation algorithm is not efficient.

The authors study the previous work of Christian Decker and Roger Wattenhofer [8] on the propagation delay in a large sample of the network nodes from the height of 180,000 and 190,000 of the public ledger; Christian Decker and Roger Wattenhofer observe the median delay time of 6.5 seconds and the mean delay time of 12.6 seconds until a node receives a block; 95% of the nodes receive a block after 40 seconds. Using the same height range, they observe the fork rate of 1.69%.

Based on the analysis, Yonatan Sompolinsky and Aviv Zohar obtain the upper and the lower thresholds of the number of processable transactions without jeopardizing the security of Bitcoin. It is shown that half of the network nodes receive a block and update their copies of the public ledger instantly since the initial announcement, while the other half needs to wait a certain amount of time; the average time it takes to deliver the data to the first half of the nodes equals to $1.80 + 0.66 \cdot$ block size.

With the given formula, the authors propose the GHOST protocol, which provides a new consensus for parent selection. Particularly, GHOST allows blocks in forks or stale blocks, and subchains to contribute to the main chain. The protocol updates the next block to be included in the main chain by choosing the block that has the most blocks appended on top of it, or the heaviest subtree, at each fork. The protocol repeats until blocks in forks are no longer present. The following algorithm is directly extracted from the white paper to describe the mechanism of GHOST. We assume the input T is the block tree and block B is the parent of the first fork in T.

Algorithm 1. GHOST [17] *Input: T*

```
1  Set $B$ as the genesis block
2  while True
3      if $B$ has no children
4          return $B$
5      Update $B$ with its child of the heaviest subtree
```

In Algorithm 1, we can observe GHOST first sets B as the genesis block and checks whether it has any children. If not, B is the only block in T, so GHOST returns B as the heaviest leading block. Otherwise, it updates B with its child that has the most blocks appended to, or the root of the heaviest subtree, and iterates until B has no children. It returns B if it is at the lowest level in T.

Finally, the authors integrate GHOST and the propagation delay to propose the lower and the upper bounds of the maximum transaction processing rate that Bitcoin can handle securely. For an optimal result, they implement lighter block that only stores transaction hashes instead of the data, by lowering the difficulty level of proof of work. They significantly decrease the size of a block and the block creation rate to accelerate transaction processing for future extensions of the model. As a result, Bitcoin can decrease the size of a block from its maximum at 1MB to 320KB and the generation interval from 10 minutes to 1 second.

3.6 Nakamoto Consensus vs. GHOST

To offer a better understanding of the selection process of Nakamoto Consensus and GHOST, the following example is demonstrated. Figure 3 shows a block tree diagram and how the two protocols differ in selecting the main chain.

In Figure 3, Nakamoto Consensus selects the chain that ends with block E, or the subtree rooted at A, because it is the longest chain it sees. It concludes with 7 stale blocks, the ones that are not included in the subtree rooted at A; the number of stale blocks is 7.

GHOST detects a fork between A and X and selects whichever that has the most blocks appended to, or the heaviest subtree. X has 6 blocks appended or the subtree rooted at X has weight 7. A has 4 blocks appended, or the subtree rooted at A has weight 5. Thus, X exceeds A in weight. GHOST adds X in the main chain, and the subtree rooted at A is no longer considered during the next selection process. The protocol repeats the same procedure for Y, T, and S that point to X. Because T has the heaviest subtree, T is added to X in the main chain. GHOST repeats the selection process for U and V. The protocol concludes with 8 stale blocks, the ones that are not included in the heaviest subtree rooted at X.

In the example, we can observe neither protocol removes the threat of selfish mining, as attackers can

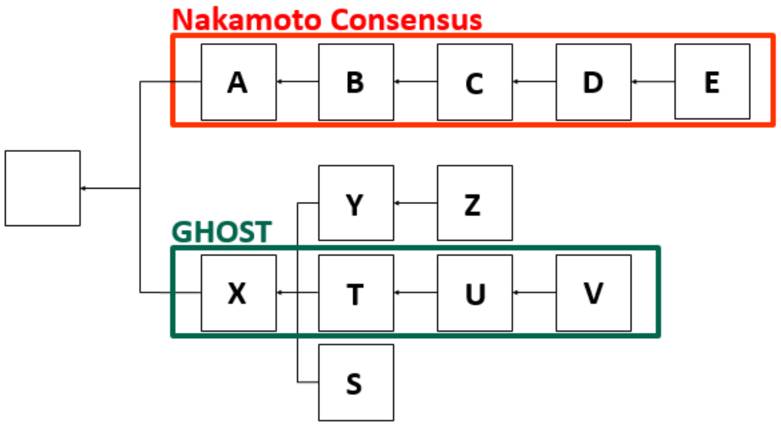

Figure 3: How the main chain is selected when multiple chains exist.

start mining on the stale block Z, V, and S in Nakamoto Consensus and the stale block E, Z, and S in GHOST for the easiest scenario. Recalling the result of Satoshi Nakamoto's experiment in Section 3.4, if the selfish miners occupy $> 50\%$ of the computational power, the probability of catching up from few blocks behind is not negligible.

4 Related Work

Ittay Eyal and Emin Gün Sirer [9] argue that our assumption on selfish mining based on %51 attack is not correct. Rather, they propose an alternative idea, a %33 attack, based on the proportion of the revenue and waste of computational resources for honest mining and selfish mining. They hypothesize the impact on the security of the Bitcoin system when the selfish miners collude and occupy $\geq \frac{1}{3}$ of the computational power over the network; the selfish miners earn more money in proportion to their pool size whereas honest miners waste more computational resources. Thus, the likelihood of honest miners to join the selfish mining pool is high.

5 Method

We continue the original analysis of GHOST and propose a new Bitcoin blockchain policy. We propose a new parent selection policy, HIRES or Highest Residual Selection, which makes nodes pick a block that has the highest residual at each fork. We test the policy with Arthur Gervais's Bitcoin network simulator [1]. We compare HIRES, GHOST, and Nakamoto Consensus under typical and selfish mining with different block parameters to optimize the efficiency and the security of Bitcoin blockchain.

Section 5.1 is divided into two parts; the first part introduces the work by Arthur Gervais et al. [10] to build the Bitcoin Network Simulator; the second part elaborates what the simulator is capable of [1]. Section 5.2 analyzes the preliminary simulation result of Nakamoto Consensus and GHOST under typical and extreme parameters. Section 5.3 is divided into four parts; in the first part, we describe our intuition on HIRES based on the analysis of the previous simulation result; the second part elaborates the algorithm of HIRES; the third part describes how we collect data for our experiments; the last part discusses how we modify the source code of the simulator and the challenges we confront.

5.1 Bitcoin Network Simulator

In Section 5.1.1, we introduce the work of Arthur Gervais et al. on building the Bitcoin Network Simulator and testing the efficiency and the security of Bitcoin blockchain using different block parameters. In Section 5.1.2, we overview the technical details of the simulator and its capability.

5.1.1 Research by Arthur Gervais et al.

Arthur Gervais et al. [10] test the correlation between the efficiency and the security of blockchains based on proof of work. They analyze four cryptocurrencies, Bitcoin, LiteCoin, DogeCoin, and to test how network parameters and real-world constraints affect the blockchain performance and security. They collect the data of each electronic payment platform from May 2015 to November 2015, specifically block size, block generation interval, the number of network nodes, and forking rate [3]. They notice Bitcoin's forking rate, or stale block rate, transcends that of other cryptocurrencies and analyze the impact of Bitcoin's block size and block generation time on the security of Bitcoin blockchain.

For its security component, they extend Markov Decision Process [16] where malicious mining activities are determined pseudorandomly to find an optimal strategy to perform double-spending and selfish mining. They follow an alternative proposal by Ittay Eyal and Emin Gun Sirer [9] on attacks to initialize the adversarial computation power to 33%, rather than 51% analyzed by Satoshi Nakamoto.

Based on the analysis, the authors build a Bitcoin network simulator that models real-world mining activities and information propagation between the network nodes. The authors point out the simulator does not model transaction propagations between the nodes as their experiment can be done with a higher level of abstraction. The simulator models 16 miners, representing 16 major mining pools of different computing power in the Bitcoin network, who perform mining activities and send blocks to nodes. The nodes, representing the Bitcoin network nodes, receive blocks and add to their copies of the blockchain. The default mode of the simulator takes the Bitcoin data and returns the statistics of each experiment, including the rate of stale blocks, the total number of blocks in each node's blockchain copy, and block propagation time. The selfish mining mode initializes 1 miner of 33 % of mining power and distributes the rest to 15 miners; the result displays the statistics of each selfish mining, including the data in the default mode plus the profit of selfish mining.

The authors run simulations to retrieve the rate blocks and the profit of adversarial mining when different block generation intervals and block sizes are given; 10000 consecutive blocks are mined in various ranges, from 25 minutes to 0.5 seconds, and their block sizes differ from 8 MB to 0.1 MB. They conclude block size and block generation time are inversely proportional; the rate of stale blocks increases exponentially as block size increases, while slower block generation interval decreases the rate; the profit of selfish mining increases as block generation interval decreases, but selfish miners are rewarded more when the block size is bigger.

5.1.2 Technical Details and Capability

As briefly mentioned in the first part, Arthur Gervais's Bitcoin Network Simulator [1] is a single-processor model with 16 threads that emulates mining activities of 16 major mining pools. Miners extend network nodes' activities during simulation, and thus miners do as much as the network nodes in propagating and validating blocks. A selfish miner can make an extra decision to mine a block, skip block propagation until his chain becomes the longest, and release it.

Each simulation takes block parameter inputs to start. If the inputs such as the size of a block, the number of nodes, or the block generation time are not given, the simulation begins with its default input of 100 blocks and 60 nodes. Upon its start, the simulator initializes a global blockchain that stores the genesis block, and each miner with different computational power performs the first mining activity. Because the simulator does not model transactions, mining a block is much simpler than what is done in reality. Each miner thread includes the height of his block, his ID, the miner ID of its parent, time of creation, time of receival, and his IP address; the miner tries to add to the current highest block in the main chain, and time of receival is later modified when other nodes receive the block. If the block size is not specified, the size of each block is drawn from by a probability distribution function based on block size intervals and weights, which reflects the real block sizes from May 2015 to November 2015[6] [10]. If block generation interval is not specified, the next block generation time is also decided accordingly to the size of the block for an optimal simulation result. Therefore, specifying block parameter inputs is very important to reflect impacts of the parameters on simulation results.

Upon its creation, each miner sends the block to its peers or the nodes whose IP addresses are connected with him. As explained in Section 3, the nodes exchange a lot of messages to check the availability of the block and announce it. We use the following example to describe communication between nodes during a simulation. In Figure 4, we visualize a network topology that represents information exchange between 300 nodes. Dots refer to nodes, and edges between dots represent information exchange between the nodes. We see miners have more edges going out and thus are darker as they initially broadcast to their peers. The nodes who receive the block repeat broadcasting until the most distant nodes receive the block. The topol-

[6]The data do not reflect block sizes as of today.

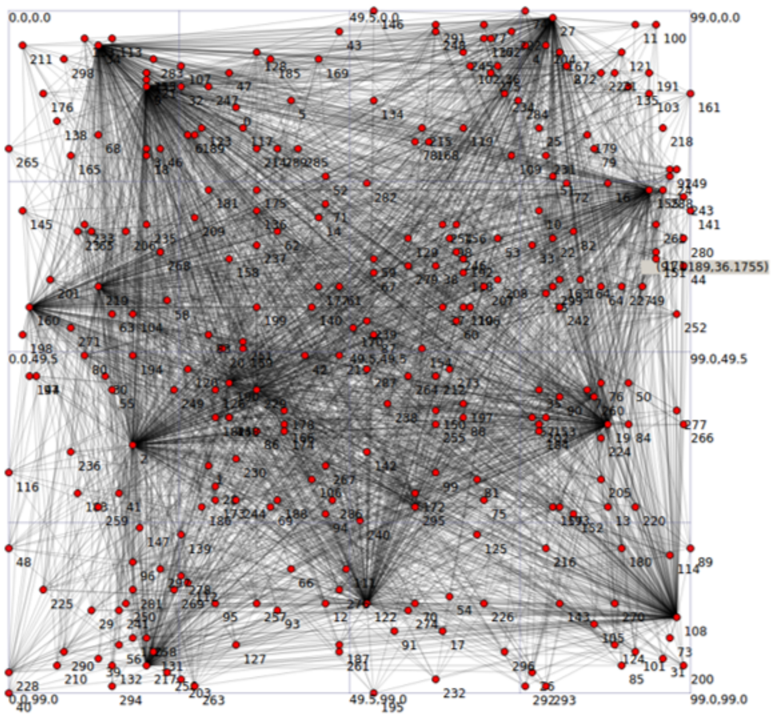

Figure 4: Information propagation between the network nodes during simulation.

ogy does not vary on different information propagation mechanisms, block sizes, the number of blocks, or block generation intervals. Rather, it reflects how miners and nodes behave during simulation.

The simulation runs until the last block of the targeted number of blocks is mined. The result of each experiment displays the statistics of the rate of stale blocks, the longest fork size, block propagation time, and blockchain copies of nodes. To thoroughly understand different block propagation delays among the nodes, consider the following example. In Figure 5, we visualize a cumulative distribution of block propagation time when passing typical block parameters. We can observe more than 80% of the nodes, or approximately 240 nodes, receive a block less than 10 seconds when the number of nodes is fixed to 300.

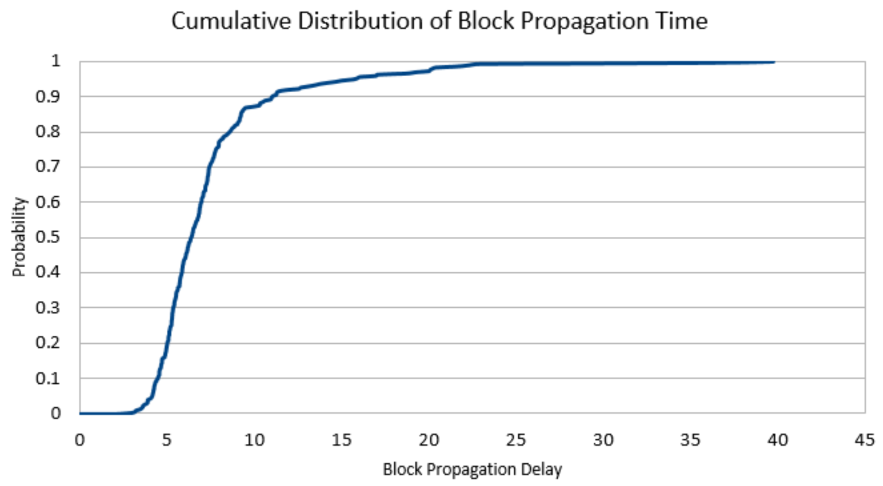

Figure 5: The cumulative distribution of block propagation time.

5.2 Preliminary Experiment: Nakamoto Consensus vs. GHOST

We conduct experiments on Nakamoto Consensus and GHOST using the simulator to discover any improvements to make. The main purpose of the simulation is to give a better understanding of why we modify the simulator and propose the alternative policy, HIRES.

For an optimal result, we remove duplicate inv message exchange for same blocks by making its timeout as long as the time it takes to run the simulation; the modification makes the nodes propagate more blocks in a fixed amount of time. The following table is presented to describe the block parameters for each protocol under honest mining and selfish mining. In Table 1, Bitcoin typically generates a 1MB block every 10 minutes [3]; Ethereum modifies GHOST to generate a 1.5KB block every 10 to 20 seconds [5]. For the extreme block parameters, we make miners to generate a 1MB block every 6 seconds for both protocols.

In honest mining test, we follow the distribution mechanism of the computational power of the source code. For each protocol, we pass each block parameter described in Table 1 along with 100 blocks and 500 nodes. Each experiment is iterated 100 times. We provide a table that describes the result of each

	Typical		Extreme	
	Block Size	Block Generation Interval	Block Size	Block Generation Interval
NAKAMOTO	1 MB	10 Min	1MB	6 sec
GHOST	1.5 KB	10 - 20 sec	1MB	6 sec

Table 1: Block parameter inputs for honest mining tests and selfish mining tests.

	Typical			Extreme		
	# Total Blocks	Stale Block (%)	Delay (s)	# Total Blocks	Stale Block (%)	Delay (s)
NAKAMOTO	94.97	1.93	23.21	57.51	68.05	93.82
GHOST	99.41	4.42	0.82	57.12	67.34	91.77

Table 2: Honest Mining Test Result 1.

experiment. In Table 2, we define # *Total Blocks* as the number of total blocks in the main chain and *Stale Block (%)* as the percentage of stale blocks in the main chain; the mean block propagation time is represented as *Delay (s)*.

In selfish mining test, we follow the analysis on 51% attack [14] and assigns the selfish miner 51% of the entire computation power. For each protocol, we pass each block parameter in Table 1 along with 100 blocks; the source code does not model nodes' activities but rather focuses on the behavior of the selfish miner. Each experiment is iterated 100 times as well. The following table is used to describe the result of each experiment. In Table 3, we define # *Total Blocks* as the number of total blocks in the main chain and *Stale Block (%)* as the percentage of stale blocks in the main chain. We call the number of blocks mined by the selfish miner # *Selfish Blocks* and # *Selfish Income* the number of # *Selfish Blocks* added to the main chain. Lastly, *Selfish Profit* is calculated as follows:

$$\frac{\textit{\# Selfish Income - \# Selfish Blocks}}{\textit{\# Selfish Blocks}} \times 100.$$

In Table 2, the simulation result of passing the extreme block parameters shows that both protocols have almost identical data. The network nodes following Nakamoto Consensus propagates 2 seconds slower than the ones following GHOST, but they slightly have more blocks in their local copies of the blockchain. We see clear differences in propagation delay from the simulation result of passing the typical block parameters; the network nodes under GHOST generates about 4 more blocks and propagates almost 30 times

	Typical				
	# Total Blocks	Stale Block (%)	# Selfish Blocks	# Selfish Income	Selfish Profit (%)
NAKAMOTO	95.74	43.09	48.28	27.12	-45.29
GHOST	92.33	41.61	49.07	35.74	-29.76
	Extreme				
	# Total Blocks	Stale Block (%)	# Selfish Blocks	# Selfish Income	Selfish Profit (%)
NAKAMOTO	97.96	43.67	49.52	34.83	-32.16
GHOST	89.51	40.29	50.17	41.85	-18.07

Table 3: Selfish Mining Test Result 1.

faster than Nakamoto Consensus. In this experiment, we conclude that the extreme block parameters are not optimal to show the efficiency of the protocols, but the typical parameters clearly demonstrate that the blockchain grows faster under GHOST than Nakamoto Consensus.

Table 3 supports our hypothesis that the blockchain with forking due to stale blocks is vulnerable to selfish mining when the selfish miners occupy more than half of the computational power. In the extreme case, the nodes following Nakamoto Consensus make 44% of the blocks in the main chain stale blocks, and the ones under GHOST let 40% of the blocks become stale blocks. However, Nakamoto Consensus is more resilient or incentivizes the attacker less than GHOST does by 14%; the attacker following Nakamoto Consensus mine about 50 blocks and adds 33% of them to the main chain while he adds 42% of them under GHOST. When passing the typical block parameters, the nodes adopting Nakamoto Consensus produce 43% of stale blocks, whereas the nodes following GHOST has 42% of stale block rate. In this experiment, we conclude Nakamoto Consensus is more resilient than GHOST as the rate of selfish blocks included in the main chain is 10% less than that of GHOST. This result also supports the analysis of Arthur Gervais et al. that block generation interval and block size is inversely proportional.

We have the simulation results that show both protocols do not satisfy the fast growth and the secureness of Bitcoin blockchain. Therefore, we hypothesize a new policy and propose modifying the simulator to test its validity.

5.3 HIRES

We propose our parent selection policy HIRES based on the analysis of the previous simulation result. HIRES extends GHOST in terms of picking a parent that is the root of the most expensive subtree, or having the highest residual value, at each fork. We discuss our intuition, algorithm, data collection, and its implementation.

5.3.1 Intuition

In the preliminary simulation result, we can observe GHOST is faster but less secure than Nakamoto Consensus. Thus, the objective of HIRES is to maintain or supersede the speed of GHOST and make it at least as resilient as or more secure than Nakamoto Consensus. We recall that miners are rewarded with newly minted coins and residuals from their block after they add it to the main chain. Our intuition is that typical miners prefer to collect pending transactions of higher residuals to generate a block with the highest residual value to maximize their block reward. If the policy is to add the block of the highest value of residuals at each fork, we hypothesize miners try their best to maximize their profits and resilience to reversion. To that extent, because the number of daily confirmed transactions are definite, once the honest miners collect the transactions, the attackers only have remaining transactions of lower residuals to collect. We suspect this behavior makes the likelihood of maintaining the honest chain higher Nakamoto Consensus because not only heavier subtrees contain higher residuals but also the attackers have fewer options. We also expect HIRES is as fast as GHOST because we assume HIRES use the same typical block parameters as GHOST.

5.3.2 Algorithm

Our algorithm of HIRES extends that of GHOST and goes five level down. To more aptly describe the selection process, we provide the diagram. Algorithm 2 describes how HIRES selects a new parent.

Algorithm 2. HIRES

```
Set the global blockchain $blockchain$
Set the initial height $height$ to $0$
Set the genesis block $B$ of $blockchain$

if $blockchain.length == 1$
    return $B$
if $blockchain.length < 6$
    $height = blockchain.length$
else
    $height = 5$

if $blockchain.(blockchain.length - height)$ is null or $> 1$
    while $blockchain.(blockchain.length - height)$ is null or $> 1$
        $height = height - 1$
        if $height == 0$
            return $B$

Set $B$ to block at $blockchain.(blockchain.length - height)$
while True
    if $B$ has no children
        return $B$
    Update $B$ with its child in the subtree containing the highest residual
```

In Algorithm 2, unlike GHOST that takes an input of block tree T, we assume a global blockchain $blockchain$ is initialized with the genesis block B. HIRES initializes $height$ to 0. If $blockchain$ only has

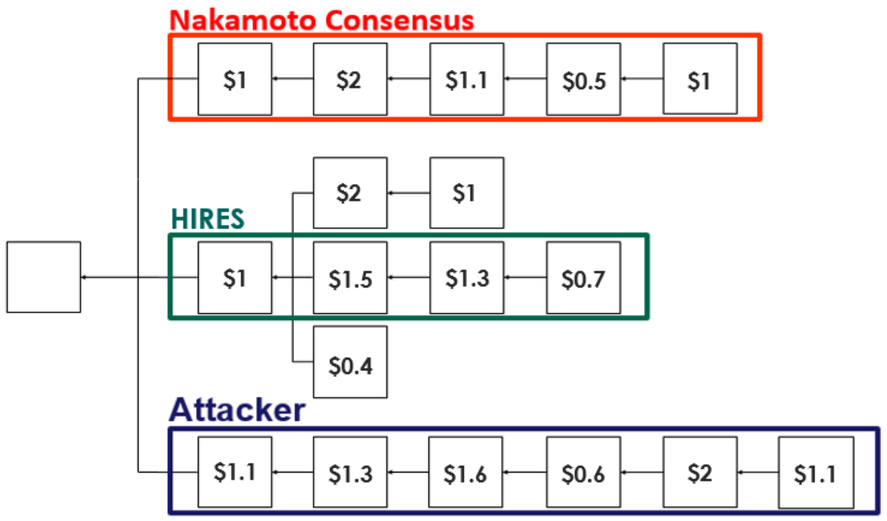

Figure 6: HIRES makes irreversible transactions.

the genesis block, then we return B; if its length is less than 6, we set $height$ to its length; otherwise, we set $height$ to 5 so that we can apply the selection process for five block heights. If $blockchain$ does not have any block or has multiple blocks at $blockchain.length - height$, we iterate to find $height$ that $blockchain$ has a single block by decrementing $height$ by 1; if $height$ is 0, then $blockchain$ only has the genesis block, so we return B. Otherwise, we update B with the block at $blockchain.length - height$ of $blockchain$ so that we go five level down. We extend the algorithm of GHOST by checking B has any children and updating B with its child that is the root of the subtree containing the highest residual. We return B if it does not have any children or is at the lowest level of $blockchain$.

We provide a diagram to describe how HIRES selects a new parent. Figure 6 compares the block diagram of HIRES and its selection process when the blocks generated by selfish mining are released at the same time. The diagram describes that Nakamoto Consensus is vulnerable to the security threat as the chain of attacker blocks can override the longest main chain. Attackers cannot override the blocks that HIRES selects as their blocks have less residuals included. We know miners keep residuals after appending their block to the main chain, so they prioritize transactions with higher residuals to include in a block.

In details, the total value of residuals of each subtree rooted at the very first forking block is \$5.6 when adopting Nakamoto Consensus, \$7.9 for HIRES, and \$7.7 for the attacker. Because HIRES selects the block rooted at its subtree as a parent and repeats its selection process within the subtree, blocks included in subtree following Nakamoto Consensus nor the ones in the attacker's chain cannot revert the block. Consequently, the attacker has to attempt mining before the parent where forking occurs, in which they have to spend more time and computing power to accomplish.

5.3.3 Data Collection

We collect transaction fee data from blockchain.info [3], the same source that Arthur Gervais et al. collect their data to build the simulator, from May 2015 to November 2015. Our purpose is to reflect the Bitcoin network in practice as closely as possible by extracting the data from the time frame used in the simulator. To obtain average transaction fees included in a block each day, we divide the total transaction fees in each day by the number of daily confirmed transactions and multiply the result with the number of transactions in a block in the corresponding day. The procedure is repeated from the first day of May 2015 until we have the data of the last date of November 2015. Finally, we shrink the data to have 28 intervals based on frequently appearing residuals and count the number of occurrences each value appears between two intervals in ascending order to obtain 27 weights. In Figure 7, we provide a probability distribution chart of transactions fees from May 2015 to November 2015.

5.3.4 Code Modification

The Bitcoin network simulator is implemented in a style of object-oriented programming. Each miner thread generates a block object that has a set of attributes, and each network node receives the object to parse and verify each attribute during validation check. Our strategy is to add a new attribute representing transaction fees to a block object so that miners can pick residuals based on the probability distribution function. Thus, every component associated with validation or mining needs to be modified. We elaborate

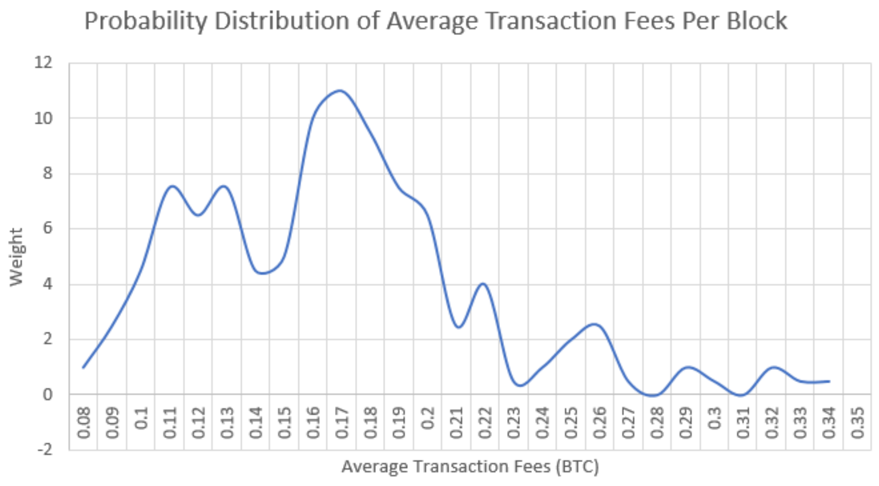

Figure 7: The probability distribution chart of transaction fees in a block.

our implementation from modifying a block object to node's activities and its challenges.

By default, a block object has 7 attributes: $blockHeight$, $minerId$, $parentBlockMinerId$, $blockSizeBytes$, $timeCreated$, $timeReceived$, and $receivedFromIpv4$ [1]. In details, when the miner generates a new block, he sets its $minerId$ to his Id. He calls the method $getCurrentTopBlock$, which returns the current highest block in his copy of the blockchain, and sets 1 higher $blockHeight$. $parentBlockMinerId$ is the previous block's $minerId$. $blockSizeBytes$ is randomly chosen from the block size distribution [10]. $timeCreated$ and $timeReceived$ are initialized to 0. Lastly, $receivedFromIpv4$ is the miner's Ipv4 address.

A global blockchain is a 2D-vector such that its row represents its height and each column at each row contains a vector of one or more block objects. After a block object is created and validated, the nodes update $timeReceived$ and call $addBlock$. The method adds a new height, or the new row in the global blockchain, along with the block object if its $blockHeight$ is previously unseen; if the same $blockHeight$ exists, then it pushes the block object to the existing row of the blockchain. Thus, we need to a new attribute, $m\_transactionFees$, in a block object for adopting HIRES. On top of that, we modify the miners to add on top of the subtree that has the highest residual.

Firstly, we initialize a global blockchain to have the genesis block of $m\_transactionFees = 0$. In details, the nodes and the miners have the same local copies of the blockchain of size 1 when the simulation starts. We incorporate the probability distribution of transaction fees from Section 5.3.3 to allow the miners pick transaction fees based on each weight between two intervals during each mining process. We confirm each block has different transaction fees, or residuals, as the attribute.

Secondly, we implement $HIRES$ based on the algorithm[7] in Section 5.3.2. We recall $HIRES$ first checks if block B has any children and updates it with its child in the subtree containing the highest residual. For checking the existence of the given block, we verify any block at 1 less $blockHeight$ such that its $parentBlockMinerId$ matches the $minerId$ of the block we are testing. Replacing the block with its child satisfying the condition of HIRES is more complex; if the block has multiple children, then we have to compare the residual of each subtree. To simplify the procedure, we use indices to retrieve our target block from the global blockchain. Specifically, we verify the subtree leading the highest residual at each fork by setting the initial residual to 0 and adding residuals recursively; if the input B, a block object, has any children, then the method calls itself for each of the children and iterates the procedure until either the last block being checked has higher height than the height of the global blockchain or it has no children. We keep the index of each subtree to select the one having the highest residual. We update B with the first block, or the root, of the subtree and iterate the process until we cannot find B's children.

Lastly, we let the miners set $blockHeight$ and $parentBlockMInerId$ according to $HIRES$. For instance, if the $HIRES$ returns a block object at $blockHeight = 15$ and $minerId = 3$, then the miner generate a new block that has $blockHeight = 16$ and $parentBlockMinerId = 3$. Based on the preliminary result in Section 5.2, we expect unsynchronized blockchain copies among the nodes, so we later modify HIRES to go five level down for an optimal simulation result.

The biggest challenge of the implementation is resolving a segmentation fault. Debugging is much more complicated as the amount of information exchange between the nodes is tremendous; when we print all the messages between 500 nodes during simulation, the line number exceeds 40 million. The first flaw we discover is that $HIRES$ does not have the nullity check; For each row in the global blockchain, $HIRES$ iterates from the first column to the last column without checking whether a block object exists. As described in Section 1.1, some nodes receive block A first while others receive block B, which leads to forking and

[7]Our initial implementation of HIRES is to start from the genesis block.

	Typical			Extreme		
	# Total Blocks	Stale Block (%)	Delay (s)	# Total Blocks	Stale Block (%)	Delay (s)
NAKAMOTO	94.97	1.93	23.21	57.51	68.05	93.82
GHOST	99.41	4.42	0.82	57.12	67.34	91.77
HIRES	98.12	4.64	0.81	58.23	68.16	93.52

Table 4: Honest Mining Test Result 2.

different state of their public ledgers. For each mining activity, the miners call $HIRES$, and some of them may not have a block object that the others have. We fix this issue by passing the nullity check every time $HIRES$ iterates rows in the blockchain, but the segmentation fault is still returned.

The second flaw we find is that the miners do not stop mining even though they receive a new block of the most expensive subtree from their peers. We fix this issue by interrupting their mining activities when a newly received block satisfies $HIRES$ and scheduling next mining activity since the interruption. After the modification, the segmentation is no longer present.

6 Simulation Result

We repeat the same strategy used in Section 5 and conduct experiments of three protocols, Nakamoto Consensus, GHOST, and HIRES. We run each experiment 100 times.

6.1 Result 1

We provide an integrated result that compares the three protocols. We analyze the experimental result in Table 4 and 5. The terminology is identically defined as in Section 5.

In Table 4, we do not see a clear difference between the three protocols when passing the extreme block parameters. HIRES propagate as long as Nakamoto, and it is 1.75 seconds slower than GHOST. When passing the typical block parameters, HIRES generates about 1 block fewer than GHOST and 0.18 more stale blocks. The average block propagation delay is almost identical. We can observe that GHOST and HIRES

	Typical				
	# Total Blocks	Stale Block (%)	# Selfish Blocks	# Selfish Income	Selfish Profit (%)
NAKAMOTO	95.74	43.09	48.28	27.12	-45.29
GHOST	92.33	41.61	49.07	35.74	-29.76
HIRES	90.92	40.31	48.27	34.63	-29.98
	Extreme				
	# Total Blocks	Stale Block (%)	# Selfish Blocks	# Selfish Income	Selfish Profit (%)
NAKAMOTO	97.96	43.67	49.52	34.83	-32.16
GHOST	89.51	40.29	50.17	41.85	-18.07
HIRES	77.62	39.31	50.16	44.59	-11.86

Table 5: Selfish Mining Test Result 2.

do not differ much during honest mining simulation.

Table 5 shows a drastic difference between HIRES and the two protocols. When the extreme parameters are given, HIRES generates the fewest blocks in the main chain, being 80% of the blocks generated under Nakamoto and 86% of the blocks in GHOST. Although it has the lowest stale block rate, it only differs by 4.4% with the protocol that has the highest stale block rate. The number of selfishly mined blocks are almost identical, but the rate included in the main chain demonstrates Nakamoto Consensus incentivizes the least for selfish mining, then GHOST, and finally HIRES. According to the result, HIRES is the most vulnerable protocol among the cluster.

The experiment result of passing the typical block parameters supports the idea that Nakamoto Consensus is the most secure among the three protocols. It incentivizes the least for selfish mining, almost letting the attacker waste the half of his work. GHOST and HIRES are almost identical in the result. Therefore, in this experiment, we analyze GHOST and HIRE are equally fast under the extreme and typical block parameters when no security threat is present; HIRES is the most vulnerable under the extreme parameters when the selfish miner has more than half of the computational power; Nakamoto is the slowest but is more resilient and more likely to let attackers lose money.

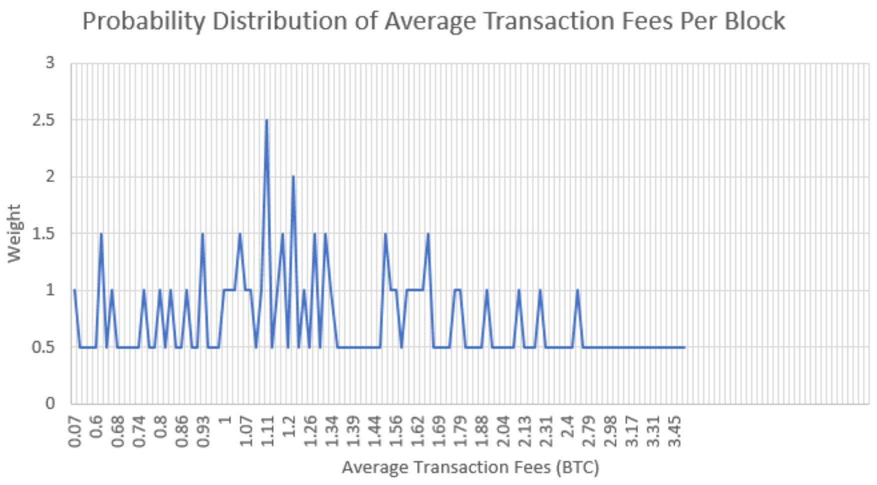

Figure 8: The optimal probability distribution chart of transaction fees in a block.

6.2 Result 1: Flaw and Solution

Based on the simulation result, we suspect the probability distribution of transaction fees is not accurate; while the source code of the simulator contains 201 intervals and 200 weights for generating the probability distribution of block sizes, we only have less than 30 transactions intervals to pick. In Figure 7, we can observe that more than 80% of transaction fees are less than 0.2 BTC; more than 60% of the blocks are likely to be less than half of the maximum transaction fee, 0.35 BTC, that can be drawn from the distribution. According to the distribution, HIRES is more vulnerable than GHOST, because if the attackers can generate blocks of residuals greater than 0.25 BTC consecutively, the likelihood of making more expensive private chain is higher. Therefore, we adjust the range of intervals and weights to draw accurate results. We redistribute residual values to make a new probability distribution composed of 201 intervals and 200 weights. The following figure is presented to show the difference between the two probability distributions used during simulation. In Figure 8, we show a more optimal probability distribution of residuals per block. We can observe that weights range from 0.5 at a minimum to 2.5 at a maximum for each residual interval.

	Typical				
	# Total Blocks	Stale Block (%)	# Selfish Blocks	# Selfish Income	Selfish Profit (%)
NAKAMOTO	95.74	43.09	48.28	27.12	-45.29
GHOST	92.33	41.61	49.07	35.74	-29.76
HIRES	94.69	40.48	50.61	36.46	-30.17
	Extreme				
	# Total Blocks	Stale Block (%)	# Selfish Blocks	# Selfish Income	Selfish Profit (%)
NAKAMOTO	97.96	43.67	49.52	34.83	-32.16
GHOST	89.51	40.29	50.17	41.85	-18.07
HIRES	88.63	35.68	49.84	41.55	-18.20

Table 6: Selfish Mining Test Result 3.

6.3 Result 2

We use the new probability distribution of residuals to draw more accurate simulation result. We reflect the previous simulation result where GHOST and HIRES are almost identical during honest mining tests. Thus, we only conduct selfish mining tests with the extreme and typical parameters. Table 6 shows the empirical data.

We see a clear difference with the data in Table 5 under the extreme block parameters where HIRES adds fewer blocks in blockchain and incentivizes the selfish-miner 7% more than the current data. When passing the extreme parameters, HIRES generates the fewest stale blocks, but it is still less secure than Nakamoto Consensus in terms of incentivizing selfish mining activity. When passing the typical block parameters, HIRES has the lowest stale block rate and incentivizes less than GHOST. The data also shows that Nakamoto Consensus is more secure than any other protocols it is being compared to.

7 Conclusion

In our experiments, the network nodes following HIRES and GHOST propagate blocks almost 30 times faster than those adopting Nakamoto Consensus, but the latter is the most resilient policy among them. HIRES generates less stale blocks than GHOST, but we cannot conclude HIRES is always more secure than GHOST because the difference in incentivizing selfish miners is minuscule. Even though our main contribution to extending GHOST to propose a policy that is as fast as GHOST and as resilient as Nakamoto Consensus is not successful in simulation, we justify the result with possible threats to validity.

First of all, we do not know whether the source code of Arthur Gervais's Bitcoin network simulator [1] does not have any flaws in design and implementation. For selfish mining, the simulator does not implement the nodes' activities, and we suspect the extension can change the simulation result. We point out Arthur Gervais et al. retrieve the block data almost 3 years ago and design the block size probability distribution in the source code. Thus, we are unable to tell whether the simulator aptly reflects the current Bitcoin environment.

Secondly, we cannot guarantee our modification of the source code does not have any flaws in design and implementation. Even though we do not get any segmentation fault, we only iterate each experiment 100 times to confirm the validity of our code. To that extent, running more experiments may trigger an internal error and pose a validity threat to the result.

Lastly, we retrieve transaction fees from May 2015 to November 2015 [3] and incorporate the probability distribution of transaction fees to derive empirical results as Arthur Gervais et al. do in their work. To more aptly test the code and derive accurate simulation results, we need to use recent Bitcoin data.

In reality, Bitcoin network is much bigger and complicated, and not only internal factors such as hashing and transaction processing but also external factors such as network bandwidth and geographic distribution of network nodes, including the block parameters we take into account during simulation, affect the performance and security of blockchain. We need a more elaborate mathematical approach to hypothesize a concrete foundation of Bitcoin blockchain policy.

8 Future Work

In future work, we would like to test the validity of the Bitcoin network simulator and our modification to test HIRES. Specifically, we want to test how aptly the simulator and our code reflect the current Bitcoin scheme when incorporating recent Bitcoin data. On top of that, we can extend the source code to model the network nodes' activities during selfish mining to observe clear differences between Nakamoto Consensus, GHOST, and HIRES in terms of the efficiency and the security of Bitcoin blockchain. Lastly, we would like to extend our work on HIRES and propose a policy based on mathematical research.

9 Acknowledgement

We would like to show our gratitude to David Frey, the system administrator at Union College, for preparing computer equipment for simulation, and to Matthew Anderson, our senior thesis advisor at Union College, for providing insight and guidance to complete our work.

References

[1] Bitcoin Network Simulator. https://arthurgervais.github.io/Bitcoin-Simulator/. Accessed on 2017-12-16.

[2] Bitcoin Wiki. https://en.bitcoin.it/wiki/. Accessed on 2018-01-13.

[3] Blockchain Info. https://blockchain.info/charts. Accessed on 2018-02-20.

[4] Cryptocurrency Market Capitalizations. https://coinmarketcap.com/exchanges/volume/24-hour/. Accessed on 2017-06-07.

[5] Etherscan. https://etherscan.io/chart/blocktime. Accessed on 2017-10-03.

[6] openbazzar. https://www.openbazaar.org/blog/new-year-new-network/. Accessed on 2018-03-13.

[7] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to better – how to make bitcoin a better currency. In *International Conference on Financial Cryptography and Data Security*, pages 399–414. Springer, 2012.

[8] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–10. IEEE, 2013.

[9] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International Conference on Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.

[10] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 3–16. ACM, 2016.

[11] Chris Hammerschmidt. Medium. https://www.medium.com/a-guide-to-dishonesty-on-pow-blockchains-when-doesdouble- spending-pays-off-4f1994074b52. Accessed on 2017-10-03.

[12] George F Hurlburt and Irena Bojanova. Bitcoin: Benefit or curse? *IT Professional*, 16(3):10–15, 2014.

[13] Panos Mourdoukoutas. Forbes. https://www.forbes.com/sites/panosmourdoukoutas/2017/03/04/bitcoin-is-better-than-gold/#7310bed95f04. Accessed on 2017-06-17.

[14] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[15] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press, 2016.

[16] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 515–532. Springer, 2016.

[17] Yonatan Sompolinsky and Aviv Zohar. Accelerating bitcoin's transaction processing. fast money grows on trees, not chains. *IACR Cryptology ePrint Archive*, 2013(881), 2013.