The Dynamics of a "Selfish Mining" Infested Bitcoin Network: How the Presence of Adversaries Can Alter the Profitability Framework of Bitcoin Mining

Contents

1 Introduction

Bitcoin is proving to be a global force: increasingly many people across the world are utilizing the cryptocurrency as a medium of exchange, and governments are starting to acknowledge its prevalence. Some argue that it has the ability to reshape economies and revolutionize how people engage in transactions. Bitcoin is made unique by its stark difference from typical currencies: it is a system that allows for the transfer of funds without the need for a physical medium or a centralized authority. Created in 2008 by "Satoshi Nakamoto," a founder who remains anonymous, Bitcoin is essentially a solution to the well-known computer science construct called the "Byzantine Generals Problem" [1].[1] In short, this problem is based on the issue of trying to reach an agreement across a large network that may not be fully reliable and could contain malicious adversaries [1]. Combining well-established public key cryptography methods, an incentivized verification system that simultaneously introduces new currency into the system (mining), and a publicly accessible record of all transactions (the blockchain), Bitcoin's efficient solution to the Byzantine's General Problem seems to make it a feasible and reliable worldwide currency.

Bitcoin's functionality as a secure currency is grounded in the efficacy of three major pillars: cryptography, bitcoin mining, and blockchain technology.[2] This paper will discuss all three pillars in depth, but the latter two are of particular focus. The first, cryptography, lays the foundation for safe transfer of funds and makes fraud extremely difficult. Public key cryptography is based on the combination of multiple well-known and dependable algorithms that make it nearly impossible to access another person's funds by attempting to guess or generate his/her "passwords" (private keys, to be precise) [2]. Cryptography is an area that is always being improved upon, especially because it is used to protect sensitive, digitally-stored information in almost any context or system. The second of our three pillars, bitcoin mining, is a truly unique self-sustaining system that achieves two important goals at once: verification of the validity of all transactions in the system and introduction of new currency into the system. Both of these goals are made possible by the incentivization of miners with

[1] "Satoshi Nakamoto" is a pseudonym for the founder of Bitcoin. No one know his/her real name.

[2] We capitalize "Bitcoin" when referring to the network as a whole or the technology as a whole; we leave it lowercase when referring to "bitcoin" as a unit of currency

bitcoin rewards. Bitcoin miners are rewarded for every block of transactions (to be discussed later in detail) they can verify, and every verified block injects a constant amount of currency into the system. The destination of said verified blocks brings us to the final of our three pillars: the blockchain. "Blockchain" is a general term for a chain of blocks of information (hence the name!), which are connected via encoded alphanumeric references to each other [1]. The blockchain is essentially a large version of the widely-used data structure, a linked list. Of course, there are specific details about blockchain, and even more so Bitcoin's blockchain, that make it more specialized than a simple linked list (we will discuss these very details later). Because a blockchain allows for efficient storage of large quantities of information, it is possible for Bitcoin's entire history of transactions, protocol changes, forking, etc. to be encoded in this data structure. Finally, it is critical to note that the blockchain is at the core of Bitcoin's success: because the blockchain is publicly-accessible, maintained locally on most Bitcoin software, and inherently difficult to manipulate, it is very hard to damage or cheat the system without it being detected by many users.

As with any new technology, there are unexpected flaws that arise as it becomes more widely used. The case is no different for Bitcoin: since its founding, there have been frequent iterations on both the system protocol and the core Bitcoin software (called Bitcoin Core) to improve upon inefficiencies and issues in the system architecture [1]. However, there are some "flaws" that are more subtle and take longer to discover than others. One that has drawn significant attention in the world of cryptocurrency-focused academicians is the incentive compatibility of bitcoin mining. Bitcoin mining, which we discussed earlier to be a verification and currency injection process, is supposed to be performed in a very specific way, guided by a set of rules known as "protocol" [8]. Because bitcoin is decentralized and no one body is "watching over" the actions of miners, it is impossible to force participants in any capacity to follow protocol. Counterintuitively, this is actually what makes Bitcoin special: the underlying technology creates a system in which breaking rules is generally not beneficial and can often be negatively impactful to the transgressor. However, in the case of mining, many scholars (beginning primarily in 2013) have found there to be potential incentive in breaking the rules. Scholars have found that following certain decision processes can lead to higher expected revenues [4; 5; 6; 3].

This paper aims to investigate one of such exceptions to the incentive-compatibility trait of the network. Specifically, we will expound upon the current literature focused on a phenomenon coined as "Selfish Mining." We will start by reviewing the status quo protocol-following mining strategy, then do a detailed walk-through of the selfish mining strategy, next build a model to predict the revenue of said miners, and finally simulate the bitcoin network with many "selfish miners" to compare their actual revenues with those that the revenue model would predict. The current literature focused on selfish mining assumes there is one colluding group that engages in selfish mining as a team—my simulations aim to investigate the more realistic case in which many miners separately engage in the subversive strategy.

Before performing the above-described modeling and analysis, it is important to discuss the technical details of the Bitcoin system. Because Bitcoin has so many unique components not well-mirrored in traditional currencies, it is necessary to build a base of knowledge before moving into the complexities of the models and simulations. Specifically, because our investigation requires a strong understanding of how the blockchain works, one must understand how bitcoin mining works; in order to understand how bitcoin mining works, one must understand the how nodes operate in the bitcoin network and how transactions are sent between them. Thus, we will start from the most fundamental aspect of the system, transactions, and build up slowly to bitcoin mining and the blockchain.

2 Bitcoin: A Technical Overview

2.1 Transactions and Ownership

We will begin our technical overview of the Bitcoin system by looking at transactions and ownership, the most fundamental aspects of Bitcoin. The transaction story we start with will set the stage for some of the more complicated details to follow.

2.1.1 A Simple Transaction

Let's begin with a simple story of transaction between two people, Alice and Bob. Alice wants to buy a \$550 painting from Bob, but she does not have the \$550 in cash with her to make the purchase. Alice offers to pay Bob in bitcoin, but Bob does not own bitcoin nor does he have a "virtual wallet" that can use to receive it. Bob goes ahead and downloads one of many available virtual wallet apps on his phone. He could have also found a website that hosts virtual wallets or downloaded a special wallet client software for his desktop computer—for convenience sake, though, he chooses to use his phone for now. As soon as Bob makes an account on the wallet (via email address, password, and potentially some other personally-identifying information), he is ready to receive bitcoin. When he opens the virtual wallet, he clicks on a button that says "receive bitcoin" which brings him to a screen with a scannable QR code (essentially a square barcode) and underneath it, a 34-character alphanumeric code. That code represents a "bitcoin address" associated with Bob. If he gives that address to Alice, she can type it into the "send bitcoin" section of her virtual wallet and indicate a number of bitcoin (or a US dollar amount that will get converted to bitcoin at the current exchange rate) she wants to send to Bob.[3] Once she hits "send," Bob's virtual wallet will now "have" the funds Alice just sent him; similarly, Alice's wallet will now have "lost" the money she sent.[4] Bob gives Alice the painting, and both sides of the transaction are complete.

[3]Note that the QR code in Bob's wallet also encodes that 34-character address. These QR codes make it easy for someone to scan with their phone rather than having to type in 34 characters.

[4]We use quotation marks with specific intention because, as we will learn, bitcoin is not stored in the same way we think of as cash or funds being stored in accounts.

Now that we have laid out the overview of a Bitcoin transaction, let's look at the technical details of what is going on under the hood.

2.1.2 Keys, Addresses, and Cryptography

The 34-character alphanumeric code that Bob's app displayed was not simply a randomly-generated code. Rather, the virtual wallet software will generate a random number and (in most cases) pass that number through the SHA-256 function, a particularly effective hashing function [1]. A hashing function is a function that generates a unique output for every input, but has an output such that there is no way to derive the input from seeing the output. In simpler terms, it serves as a one-way function that can only be used to produce outputs such that one can't ever derive inputs from outputs [1].[5] After passing the random number through the SHA-256 function, the wallet software will store that resulting output as a "private key." A private key is the most critical component to anyone's ownership of bitcoin: it's like a password in the sense that anyone who has it can gain ownership of that person's bitcoins, and if it's lost, the owner can never access his/her bitcoin again. This is an important point worth emphasizing: private keys are the only way one can access his/her bitcoins, and anyone who obtains private keys has the rights to bitcoins associated with those keys. We will return to private keys shortly. For every private key a wallet generates, it also generates a "public key" using a special process called the Elliptic Curve Digital Signature Algorithm (ECDSA), which essentially uses the geometry of an Ellpic Curve to generate a corresponding string of alphanumeric characters that is uniquely associated with the input private key [1; 7]. As with a hashing function, person A cannot "back out" a private key that belongs to person B with just the public key of person B—the ECDSA is one of the most secure algorithms in terms of concealing input information [7]. Now that the wallet software has generated a corresponding pair of keys, a private key and a public key, the software can generate a bitcoin address.

A bitcoin address is created by passing a public key into a few hashing functions (in

[5]This leads to an important consequence: it is simple and quick to verify that a certain input generates an output that someone claims it does, but seeing the output alone in no way reveals the input. This fact will become critical in our section on bitcoin mining.

`123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz`

Figure 1: This is the set of characters often referred to as "Base58," which make up bitcoin addresses . Note that the "l" (lower case L), "I" (upper case i), "O" (upper case o) and "0" (zero) are left out of the character set to avoid potential confusion or mis-copying of addresses due to similar-looking characters [1].

a similar way the private key was created via hashing a random number), and the output will be a string between 26 and 35 characters [9]. Specifically, the output string with almost always be in "Base58," meaning each character comes from the set of characters shown in Figure 1.

A bitcoin address can encode a few different pieces of information and serve multiple purposes, but we will focus on its most fundamental and relevant use. All bitcoin transactions involve at least two bitcoin addresses: an address of the sender and an address of the receiver.[6] These addresses associate transactions with an "owner" in the sense that only someone with knowledge of the private key associated with a bitcoin address can affirm a transaction involving that address. Furthermore, only holders of the private key associated with a bitcoin addresses have ownership of the funds associated with that address. One might ask: How can someone in the network confirm that a person claiming to be associated with a bitcoin address is truly in ownership of that address? Well, any bitcoin software (be it a virtual wallet or otherwise) that has the ability to transact bitcoin also has the ability to "check" if a private key and public key pair match a given bitcoin address [1]. We can refer to our to our earlier discussion of the transaction between Alice and Bob to explain this point in detail.

In Bob's transaction with Alice, when Alice hits "send" on her virtual wallet, the virtual wallet is creating a transaction (which has a lot of other information in addition to the amount of money being transacted, the address of Alice and that of Bob, and a time stamp of the transaction) with a unique "signature" produced from a hashing of her private key and her public key.[7] A "signature," also a string of characters, is something that can be

[6]The one exception to this is what's called a "coinbase" transaction, which contains a receiving address of a bitcoin miner but no sender address. This is the special transaction that awards bitcoin miners: we will discuss the purpose and mechanism of rewarding miners in section 2.3.2.

[7]When we say "a hashing," we refer to putting some input through a hashing function to get an encoded result of the input.

confirmed by anyone to be valid, but does not give away Alice's private key [8; 9]. That is, someone with Alice's bitcoin address (which is a public piece of information) and a signature from one of her transactions (which Alice's software created by combining her private key with her public key) can confirm that Alice in fact (or someone possessing her private key) initiated that transaction. Someone trying to forge Alice's identity without her private key could attempt to create a fake transaction using her (public) bitcoin address, but anyone could check with their software that the signature produced from the forged key and the (public) bitcoin address is not valid [2]. It is this very idea of a non-forgeable signature which makes fraud in bitcoin nearly impossible. The computer science principles behind public key cryptography are constantly improving ahead of adversaries, and it is the very component of the system that is likely to keep rampant fraud out of the bitcoin network [2].

2.1.3 Ownership

One of the most difficult concepts underlying bitcoin is what it means to "own" bitcoin. Contrary to common belief, there is no such thing as an "account" in Bitcoin. Virtual wallets are misleading, because they convey the idea that one's money is "stored" in an account of sorts. It turns out that Bitcoin is made unique by its lack of a traditional ownership structure.

One's ownership of bitcoin is defined by any transaction in Bitcoin's history (which is encoded in the blockchain, to be discussed in section 2.4) that involves a bitcoin address to which someone has the corresponding private key [8]. So long as someone can use any bitcoin software and provide a valid private key for one or more bitcoin addresses that have been used in past transactions, he/she can then sign off on sending bitcoin earned in those transactions to other bitcoin addresses [1] . Thus, ownership is as simple as that: having a private key that produces a valid signature with a bitcoin address gives someone "ownership" of all funds pledged in past transactions to that bitcoin address [1; 2; 9]. And thus, funds are not stored in a virtual wallet or an account—rather, funds for a given user are found in all blocks on the blockchain that contain transactions involving any of that user's bitcoin addresses. Furthermore, for some bitcoin owner A, gaining or losing funds is a function of transactions being added to the blockchain that either pledge bitcoin from another bitcoin address to one of A's bitcoin addresses or pledge bitcoin from one of A's bitcoin addresses to another bitcoin address.[8]

Another important question that might arise is: Where does the bitcoin come from when someone "buys" their first bitcoin? If someone wants to purchase bitcoin with another currency (US dollars), he/she has a few options. Note that all options, whether it is putting a credit card number into an online virtual wallet to purchase bitcoin, giving cash to a friend for bitcoin, or going to a specialized bitcoin ATM, all methods share the same trait: all bitcoin are coming from other past transactions. So, when someone types a credit card number into a virtual wallet online, the credit card company actually has ownership, via one or more private keys, of bitcoin that it then transfers to the user in exchange for US dollars or another standard currency. In the case of a bitcoin ATM, a user inserts cash into a machine, and that machine also has private keys associated with bitcoin addresses, thus creating a transaction to the user's bitcoin address [1]. In all of these cases, bitcoin is never being created out of thin air—it is always coming from another bitcoin address that has been involved in past transactions [8]. [9]

One final note about pairs of bitcoin addresses is that a user can create as many of them as he/she would like. Because there are about 2^{160} possible bitcoin addresses, users can utilize their software to create as many or as few as they'd like, each with a corresponding private key [9]. Similarly, a user can decide to use more than one of his/her bitcoin addresses in a single transaction (whether sending or receiving).

2.1.4 Transactions: Inputs and Outputs

In our final section on "Transactions and Ownership," we look at the composition of a transaction in more detail. All transactions have at least one input and one output, but oftentimes have more than one input or output [1; 8]. Let's use the Alice and Bob transaction story as a starting point for this discussion.

[8]The process by which transactions get added to the blockchain is bitcoin mining, to be discussed in 2.3.

[9]As mentioned in an earlier footnote, the one exception to this rule is the "coinbase" transaction which awards a bitcoin miner using bitcoin essentially created out of thin air.

When Alice hits “send” on her virtual wallet, the software is “gathering” the necessary funds for the transaction under the hood. Virtual wallets have varying levels of sophistication, but they all have the ability, in some way or another, to obtain past transactions associated with the bitcoin address(es) that Alice is using to pay Bob. The virtual wallet must gather enough past transactions that involve that (those) bitcoin address(es) such that the sum of receipts from those past transactions sum to at least \$550 worth of bitcoin (the amount that Bob is charging Alice for the painting). Let’s say the software is able to gather three separate transactions that pledged \$100, \$200, and \$300 (worth of bitcoin), respectively, to one of Alice’s bitcoin addresses. The wallet will then construct a transaction by using the outputs (the three amounts listed) of past transactions as inputs into the transaction to Bob. These outputs from past transactions, which Alice has not yet spent (but is about to), are often referred to as UTXOs, or “Unspent Transaction Outputs” [1]. What one might notice, however, is that the sum of the inputs is \$600, more than the \$550 required. Because bitcoin can never be “split” into fractions without the use of a transaction to do so [8], the software will actually create two outputs in the transaction to Bob: one will pledge \$550 to Bob’s bitcoin address, and the other will pledge the remaining excess \$50 dollars (back) to Alice’s bitcoin address. This is how non-exact amounts from past transactions feed into future transactions: unless there is an exact match between past output amounts and the required input amount, there will always be an additional output in a transaction that sends the remaining bitcoin back to the address of the sender.

There is one more key aspect of transactions that we have not discussed yet: outputs can add up to less than the inputs. When this is the case, the leftover amount is an implied transaction fee [1]. That is,

$$\text{Transaction Fee} = \text{Inputs} - \text{Outputs}. \tag{1}$$

Transactions fees will become relevant later when we discuss bitcoin mining. A transaction fee is essentially a “tip” for the miner who ends up mining a block that includes this transaction [8; 1; 13]. In summary, inputs and outputs for a transaction can be thought of

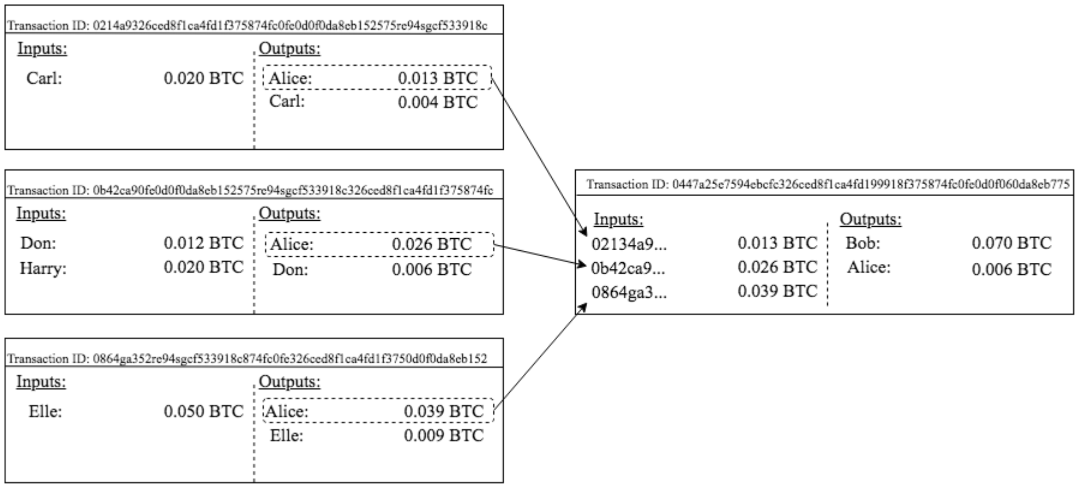

Transaction ID: 0214a9326ced8f1ca4fd1f375874fc0fe0d0f0da8eb152575re94sgcf533918c
Inputs:
Carl: 0.020 BTC
Outputs:
Alice: 0.013 BTC
Carl: 0.004 BTC
Transaction ID: 0b42ca90fe0d0f0da8eb152575re94sgcf533918c326ced8f1ca4fd1f375874fc
Inputs:
Don: 0.012 BTC
Harry: 0.020 BTC
Outputs:
Alice: 0.026 BTC
Don: 0.006 BTC
Transaction ID: 0864ga352re94sgcf533918c874fc0fe326ced8f1ca4fd1f3750d0f0da8eb152
Inputs:
Elle: 0.050 BTC
Outputs:
Alice: 0.039 BTC
Elle: 0.009 BTC
Transaction ID: 0447a25e7594ebcfc326ced8f1ca4fd199918f375874fc0fe0d0f060da8eb775
Inputs:
02134a9... 0.013 BTC
0b42ca9... 0.026 BTC
0864ga3... 0.039 BTC
Outputs:
Bob: 0.070 BTC
Alice: 0.006 BTC

Figure 2: All transactions have inputs that come from the outputs of previous transactions, as depicted above. Because of the input-output structure of transactions, chains can extend to millions of connected transactions.

like double-entry bookkeeping: every transaction has a number of inputs (debits) and a number of outputs (credits), while the difference between them is the transaction fee [1].

Let's end by visualizing what one link in a chain of transactions might look like. Using the exchange rate at the time of this paper's writing, we convert the earlier discussed dollar amounts ($100, $200, and $300) into bitcoin, and we analyze the three past transactions (pledging money to Alice) that feed into Alice's transaction with Bob.[10] We use Figure 2 to visualize this situation. The three rectangles on the left are all transactions somewhere on the publicly-available blockchain in which people sent bitcoin to Alice. The words "Carl," "Don," "Henry," and "Elle" are written in the inputs column only to illustrate that money pledged in past transactions to Alice may have come from a variety of people— however, these names actually represent entire past transactions associated with those people, in which they earned the funds that they then paid to Alice. In the rectangle to the right, we use the beginning of long alphanumeric codes (rather than names) in the inputs column to reference the transaction IDs of the past transactions (namely, the three shown on the left) that are feeding into Alice's transaction with Bob. The outputs of the three transactions on

[10]This depiction contains many simplifications, but the overall structure gives a good base of how a transaction appears under the hood.

the left don't add up to the inputs—as we discussed, this difference is implicitly a transaction fee that will later get awarded to the miner who mines a block including this transaction.

One might ask how the virtual wallet chooses which past transactions to include as funds for a new transaction—the answer is that each software does it differently, but what's important is that it is able to gather enough transactions of sufficient value to meet the necessary input requirements of the new transaction (assuming the sender actually has the required funds) [1]. As one can imagine, transactions can build up into huge chains, each transaction referencing and being referenced by other transactions. These transaction chains get bundled into blocks (which contain these longs transaction chains among other information and metadata), and those blocks get chained into a long chain of blocks (called the blockchain!).

Now that we have discussed transactions and ownership, we are ready to talk about the Bitcoin peer-to-peer network.

2.2 A Peer-to-Peer Network of Nodes

One of the factors that allows the different components of the Bitcoin system to work so well and so quickly across thousands of software clients is its peer-to-peer system ("P2P," as it is often called) [8]. One book about Bitcoin sums up peer-to-peer well: "Peer-to-peer, or P2P, means that the computers that participate in the network are peers to each other, that they are all equal, that there are no 'special' nodes, and that all nodes share the burden of providing network services" [1].

2.2.1 Nodes

Earlier we discussed virtual wallets, which give users the ability to transact bitcoin, among other capabilities. Virtual wallets turn out to be one of many forms of what are called "nodes" on the bitcoin P2P network. In fact, throughout this paper, we will very often refer to the term "Bitcoin network" (or "Bitcoin System"), which refers to all of the nodes in the network that run the peer-to-peer bitcoin protocol [1]. On the most basic level, a "node" is software the connects to the internet in some way, almost always communicating with other nodes [1]. In the case of Bitcoin, there are four major functions/traits a node tends to have: wallet capabilities, storage of the entire blockchain, mining functionality, and connection to other nodes in the network [1]. Nodes that exhibit these four capabilities are called "full nodes" [1]. Back in 2008, when Satoshi Nakamoto created Bitcoin, he also created "Bitcoin Core," the original software that served as the first and only full node software. Since then, other software applications have been created for specific purposes (i.e. consumer virtual wallets, high-powered bitcoin mining software, etc.), but Nakamoto's original software is the base from which essentially all Bitcoin-related software stems. Some nodes (i.e. those of a full-time bitcoin miner) may be better-equipped in one of the four categories and worse-equipped in others. However, just about all nodes have some way of connecting to other nodes, even if it's not via the traditional P2P protocol: they may connect to other nodes via the "stratum" protocol, for example, which is another connection mechanism that is designed for more efficient bitcoin mining [1].

Furthermore, it should be noted that all transactions in Bitcoin that end up getting recorded to the blockchain originate at some node—that is, anything that occurs involving movement of bitcoin must occur via nodes in the P2P network. Any transactions agreed upon offline or without the use of some software that connects to nodes via P2P protocol are not official until they get into the P2P network.

2.2.2 Spread of Information

Nodes in the network, full or otherwise, are connected to some number of other nodes in the network, chosen more-or-less randomly by each software [1]. The P2P protocol generally determines which nodes a given node is connected to, but which nodes a node is connected to turns out to be irrelevant: because all nodes have anywhere from a few to a few hundred connections, a piece of information can spread from one node to any other node in a matter of seconds. This mechanism of information spreading to thousands of nodes within a matter of seconds is called "flooding" [1].

The speed at which information moves between nodes in the network, referred to as "latency," is a very important aspect of the Bitcoin P2P network. Fast propagation of information like transactions and recommended changes to the Bitcoin protocol are what make the Bitcoin system efficient. Fast propagation also means that information is seen by many different people in a short amount of time, making the system less susceptible to loss or malfunction. For example, if someone's virtual wallet permanently crashes, it is not as if all of their transactions are gone: because of the movement of information to just about every other node in the network, valid transactions can basically never be lost.

In our story of Alice and Bob, when Bob downloaded his virtual wallet, it was building connections to nodes in the network over wi-fi under the hood, unbeknownst to Bob. Furthermore, when Bob received confirmation of the transaction from Alice on his wallet, that transaction notification may have come from another node that received Alice's transaction before Bob's did. In other words, it's not as if when Alice hit "send" on her wallet, that it credited Bob's wallet directly. Rather, a detailed transaction was created and broadcasted by Alice's wallet (which is ultimately just one of the many nodes in the P2P network) to the rest of the network, and it quickly arrived at Bob's wallet (also a node) via flooding [1].

Armed with the intuition for transactions and the peer-to-peer network, we have the tools necessary to discuss bitcoin mining.

2.3 Bitcoin Mining

Bitcoin mining is the process that allows transactions to be quickly processed and verified in the Bitcoin system. Mining is the mechanism that ultimately keeps the Bitcoin system running safely without the need for a centralized body; mining makes decentralization of a currency possible.

2.3.1 Purpose of Mining

With the information we have reviewed so far, let's explain why mining is a required function of the system. With cryptographically secure transactions and a P2P network that shares information about transactions, what else does the system need to work properly? Well, one aspect that's absent is an *organized* way to store all of the transactions that are being sent around. Similarly, there has to be some agreed-upon record of what transactions have been confirmed and validated; otherwise, there could be disagreement about whether the inputs to a certain transaction are fully valid. As soon as there is disagreement about one input being valid, that can put a whole chain of transactions into question—as one can guess, this problem gets out of hand quickly if not kept under control. When Bob gives his painting to Alice, he is trusting that he now owns the rights to the money she has sent him in bitcoin; but, unless his virtual wallet can tap into an agreed upon and publicly accessible record of transactions quickly, he could never be sure he has actually been paid properly. How can the network, with so many moving parts, reach a consensus and subsequently distribute that consensus to all participants? This question brings us to the first problem that mining solves: organization and verification of large volumes of transactions occurring in the Bitcoin network.

The second problem that mining solves is directly related to the term "mining." Like valuable precious metals (i.e. Gold) that act as a store of value but cannot be printed like money, bitcoin mirrors this type of commodity [1; 8]. Just as Gold is slowly mined over time and is essentially finite, bitcoin is also mined over time and is definitely finite. One difference between these two currencies is that bitcoin is mined in fixed intervals, with an already-set timeline of creation; Gold, of course, can be discovered in spurts and it is never clear how much more will be discovered. This means that bitcoin is what one might call a "deflationary currency" [1; 10]. As the demand for bitcoin goes up, but the amount of currency in the system does not increase at the same rate, each bitcoin becomes more valuable. This raises the price, and thus, over time, more dollars (or another currency) are required to purchase the same amount of bitcoin. We will discuss how exactly mining actually introduces new bitcoin into the system—but, on a high level, mining acts as a steady introduction of fresh currency into the network over time.

2.3.2 The Mining Process and Proof-of-Work

We know from our discussion of P2P that bitcoin transactions are constantly being sent between nodes. Miners, who are part of the P2P network described earlier, will start the mining process by using their software to collect transactions that come in from other nodes. A node's software is able to go through each transaction and check whether the signatures involved are valid—invalid transactions are immediately disregarded and don't make it any further in the mining process [1]. We have already built the intuition for how the software can check transaction validity—it checks all of the signatures involved, making sure each was created with a proper private key. Next, "the node verifies every transaction against a long checklist of criteria" [1]. This process ensures that the integrity of all aspects of each transaction is confirmed. Next, the node will collect these transactions into a pool of transactions [1]. Once a miner is ready to begin the next step of mining, his node will collect transactions from his transaction pool into a list of transactions (a block) such that the maximum size of the block is no more than 1 megabyte: 1 megabyte happens to be the current maximum block size in Bitcoin [1].[11]

Now that the miner has a "block" of transactions prepared, he/she will begin the main step of the mining process. In this step, the miner has a task: he/she will try to to find some number, called a "nonce," which when passed through a hashing function (SHA-256, the same as discussed earlier) with the header of the block, returns a value below a certain threshold [1]. The header of a block is essentially a set of fields of information that encodes a block's metadata and the transactions in that block—it also contains an alphanumeric code that represents a hash of the previous block on the blockchain upon which this new block may follow.[12] So, the miner will pass this header of information and a random number (the nonce) through a hashing function, hoping that the output will represent a number below a certain threshold [1]. The SHA-256 function has 2^{256} possible outputs, and each possible output can be thought of as a number from 0 to 2^{256} [9]. In order for the miner to successfully "mine" this block, he/she must find a random number (nonce) that, when hashed with the block header, outputs a value that represents a very small range within the

[11]In this paper, references to a "node," and a "node's software" have equivalent meaning.

[12]In simple terms, a hash of a previous block in the blockchain is essentially all of the information of a block passed through a hashing function (like SHA-256) such that a string of alphanumeric characters is returned—this alphanumeric code is the block's hash. As we will see, all mined blocks are "mined" on top of a block that is already in the public blockchain—thus, every block must contain information (like a hash) of the block that comes before it.

interval $[0, 2^{256})$.[13] To give context to the size of this range, 2^{256} is just under the estimated number of atoms in the observable universe [11]. Thus, the time it takes for a miner to find a nonce that outputs a below-threshold value is a direct function of how fast his/her software can generate and attempt different random values.

In order for a miner to even have a chance at successfully mining a block, he/she must invest in specialized hardware that gives his/her software more power to generate numbers more quickly. This power is called "hashing power."[14] When and if a miner successfully finds a nonce that generates a hash value below a set threshold, he/she can now broadcast this block to other nodes in the network, along with the nonce that he/she used to successfully mine the block [8]. One might ask: How can another node tell that a block it is receiving is valid? This is where the usefulness of hashing functions becomes relevant as it did earlier in our discussion of cryptography: the correct nonce to mine a block is extremely time-intensive to find, but confirming that a given nonce is correct takes a trivial amount of time (i.e. a few milliseconds). Once nodes receive this miner's block along with the nonce, they can quickly check that the block was in fact successfully mined.[15] The fact that the miner provides the nonce serves as what is called "proof-of-work." Because the only way to find a correct nonce to mine a block requires copious amounts of hashing power and time, a correct nonce is literally "proof" that the miner likely needed to perform an intensive amount of "work" to successfully find a working nonce, hence the term "proof-of-work" [8].

If miners want to increase their rate of mining blocks, they have to increase their hashing power—this costs money, of course, because more hashing power requires more hardware and more energy. Thus, a miner must carefully consider his/her expected profitability from mining when deciding how much money to invest into the process. This is where the incentivization of bitcoin mining comes into play, mentioned briefly in the Introduction section.

[13]The value (threshold) below which the miner must generate is often referred to as the "target" value.

[14]For every nonce a miner can generate, he/she can generate a hash of that nonce (together with the block) using a hashing function. Thus, hashing power, also called "hashing rate," is the number of hashes that can be computed in one second. For example, 1 MH/s $= 1{,}000{,}000$ hashes computed per second.

[15]Once a miner successfully mines a block, he/she necessarily writes the nonce into one of the information fields of the block, so that other miners receiving the block can confirm that the block was successfully mined.

Blocks provide two sources of revenue to a miner: 1) The block reward, and 2) transaction fees [1; 8].[16] The block reward is a fixed amount of bitcoin (currently at 12.5 BTC) that a miner can claim upon mining a block successfully [8].[17] How a miner does this is they create a special transaction, called a "coinbase" transaction, in the block that they mine that awards 12.5 bitcoin to a bitcoin address of his/her choosing. Unlike every other bitcoin transaction, block rewards do not have inputs from other transactions: this is the new currency that is "mined," as Gold is when a new supply is discovered. Thus, when a block is mined, 12.5 BTC is not only rewarded to the miner, but there then exist 12.5 BTC in the system that was not there before. This block reward changes by a factor of $\frac{1}{2}$ approximately every two years: it started at 50 BTC when bitcoin was founded and is slated to go to 0 in 2040 [1; 8]. This halving is intentional, so that the injection of currency into the system is slower and slower over time, eventually going to zero.

We now transition to the other source of income for miners: transaction fees. As discussed earlier, the difference between outputs and inputs in every transaction is a transaction fee. When a miner mines a block successfully, all of the transactions fees in each transaction in that block will be pledged to a bitcoin address of the miner's choosing.[18]

In summary, a miner is incentivized to invest in mining hardware because of the block reward and transaction fees associated with every block he/she can mine. As bitcoin becomes more popular (and demand rises), its price rises. When the price of bitcoin rises in terms of, say, US Dollars, the block reward and transaction fees become more valuable. This causes miners to invest in more hardware, which in turn means there is more hashing power in the bitcoin system. More hashing power means the higher volume of transactions from the higher bitcoin demand can be handled. It is this very cycle that causes bitcoin mining to be self-sustaining—it is a well-designed system that serves both miners and users of Bitcoin.

[16]See Equation 1 for a reminder about transaction fees.

[17]As we will discuss in subsequent sections, a mined block must end up on the main branch of the blockchain in order for its miner to receive the block reward and transaction fees. Most mined blocks do in fact end up on the main blockchain branch, but there are cases in which they don't—it turns out this concept is critical to this paper, as we will see in subsequent sections.

[18]One of the main reasons that a transactor would even include a "tip" for a miner as a transaction fee has a complicated reason, outside the scope of this paper.

2.3.3 Difficulty and Block Mining Rates

We mentioned the “threshold” or “target” value below which a miner has to hash a block header and a nonce to successfully mine a block. Moving this target up or down would change the difficulty of the task, and as it turns out, adjustments to the target are made approximately every two weeks in the bitcoin network. “Difficulty,” the term often used to formally describe the difficulty of hashing below the target, is readjusted so that the average time between blocks being mined is 10 minutes [1]. Every 2016 blocks that are mined, nodes on the network compare how long it took for those 2016 blocks to be discovered (using the time stamps written in each block’s header) to the following value [1; 9]:

$$2016 \text{ blocks} * \ 10 \ \frac{minutes}{block} = 20,160 \text{ minutes.} \tag{2}$$

Thus, if the previous 2016 blocks were not found in 20, 160 minutes, then the target is adjusted such that the expected time it takes for a block to be mined returns to 10 minutes—this also means the expected time for the discovery of the subsequent 2016 blocks is 20, 160 minutes.

Because Bitcoin was designed such that currency is introduced into the network at a very steady and predictable rate, it was an intentional choice to force the block discovery rate to be 1 block every 10 minutes. And because there are always new miners joining the network, and some existing miners who are purchasing additional equipment to increase their hashing power, the rate of discovery would rise over time without the aforementioned adjustments.[19] Because of the way that blocks are mined, via calculating many hashes, each with an extremely low probability of being correct, a “Poisson process,” a well-know stochastic process, can be used to model the timeline of block mining/discovery [12; 6; 5].

Because it plays an important role in the simulation portion of this paper paper, we will overview the attributes of a Poisson process. In Figure 3 we can use a line to represent a timeline, on which moving rightwards equates with moving forward in time; moving leftwards represents moving backwards in time. The line in Figure 3 represents 1

[19]If more hashing power throughout the network is going towards the mining process, correct nonces would be found faster, leading to more blocks discovered at a quicker rate.

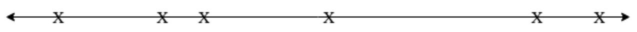


Figure 3: This represents a theoretical 1 hour in the bitcoin network, in which each X represents the time at which a block was mined by someone in the network.

hour in time in which six blocks were mined—the "X"s on the line represent times at which each of the six blocks were mined successfully by someone in the network. A Poisson process has two particularly useful and special properties that are relevant in the context of bitcoin mining and the growth of the blockchain over time. These properties will be of particular importance when we arrive at the simulation portion of this paper.

Property 1: The number of blocks discovered over an interval of time, t, denoted as N_t, in which blocks are discovered at a rate of λ, is distributed as follows[20]:

$$N_t \sim \text{Pois}(\lambda t) \tag{3}$$

and the probability density function for some number of k blocks for a Pois(λt) random variable is as follows [12]:

$$P(N_t = k) = \frac{e^{-\lambda t}(\lambda t)^k}{k!} \tag{4}$$

Property 2: The time between the discovery of any two consecutive blocks is distributed exponentially, with parameter λ, the rate of block discovery via mining. Formally, if we let T_i be the i^{th} block mined, then we can assert that [21]:

$$T_{i+1} - T_i \sim \text{Expo}(\lambda) \tag{5}$$

where the probability density function for $T_{i+1} - T_i$ being equal to the some time interval length, t, is given as:

$$P(T_{i+1} - T_i = t) = \lambda e^{-\lambda t} \tag{6}$$

[20]Pois stands for the Poisson distribution
[21]Expo stands for the Exponential distribution

Property 1 allows us to predict how many blocks will be mined, corresponding with the “X”s in Figure 3, in a given interval of time in the bitcoin network. Property 2 allows us to understand traits of the time intervals between each mined block, which correspond with the spaces between each “X” in Figure 3. Furthermore, Property 2 has an important implication which stems from the “memoryless” property of the exponential distribution: even if it has been a long time since the last block was mined (i.e. well over 10 minutes), it is no more or less likely that a new block will be mined soon [12].

When the nodes in network readjust the difficult of mining as discussed earlier, these nodes are essentially using the known amount of hashing power in the network and a target λ (and equivalently a target N_t), such that the the probability distribution function in both equation 4 and 6 stay the same over time. In the same vein, because the target rate of block discovery is 1 block every 10 minutes, the following expectation needs to stay constant over time:[22]

$$E(T_{i+1} - T_i = t) = 10 \tag{7}$$

minutes between block discovery events.

And because the expectation of an exponentially distributed random variable is $\frac{1}{\lambda}$, then we can say the rate, λ, in Equations 4 through 6 is 1/10, assuming that everything is measured in the minutes [12]. Thus, we have shown that block mining in bitcoin can be represented by a Poisson process with rate $\lambda = 1/10$. To be clear, this result is not new, and I do not take credit for discovering it: in the academic bitcoin community, the Poisson nature of block discovery is well-known [8; 6; 5; 4; 3]. This Poisson process information will be very relevant to our simulation of the bitcoin network in Section 4.

[22]The notation E(X) is meant to represent the expectation of the random variable X

2.4 The Blockchain

2.4.1 Appending the Blockchain

Once a miner mines a block successfully and broadcasts it to other nodes in the network, what happens next? Most nodes maintain an electronic copy of a chain of valid, mined blocks from the first ever mined block (called the genesis block, created in 2009)—this chain is called the blockchain, and it gets appended with new blocks when miners mine blocks [1; 8]. Currently, the blockchain has over $515,000$ blocks on its main chain, and it grows with every newly-mined block. We use the phrase "main chain" because, like a tree, the blockchain has many branches (typically called "forks") which contain validly-mined blocks [16; 3]. However, by the Bitcoin protocol, the only official chain of blocks is the one that is the longest [1; 8]. This is called the main chain. "Height" is the term used to refer to the number of blocks that were mined before any particular block on the main chain. Thus, in the case of the current Bitcoin blockchain, there many more than $515,000$ total blocks in the blockchain due to forks (branches), but there are only that many on the main chain—and thus, we say that the height of the blockchain is currently around $515,000$.

When a node receives a block that has been broadcasted by a miner, after verifying its proof-of-work, the node adds that block to its local copy of the blockchain [1; 9]. Nodes receive blocks at different times, and thus nodes may have slightly different versions of the blockchain at any given moment. These differences get resolved over time, however, because nodes are constantly getting the most up-to-date blocks from other nodes in the network. The P2P network, and its fast propagation of new information to all nodes in the network, is designed to prevent long-term differences in the blockchain between nodes. To describe why having an out-of-date version of the blockchain can hurt a miner, let's use Figure 4.

In the Figure 4, we represent a small portion of the blockchain (we show just a piece of the main chain) from the perspective of two different miners, miner A and miner B. Miner B is well-connected to the P2P network and has the most up-to-date version of the blockchain. Unbeknownst to Miner A, his node has been experiencing some internet trouble, and he has thus not been able to receive blocks for the last twenty minutes from the

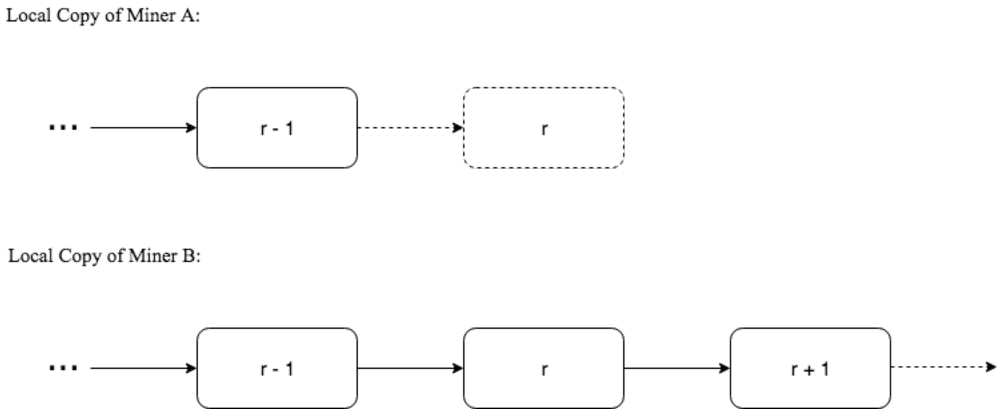


Figure 4: Miner A has a copy of the blockchain that is behind that of miner B, which means that if miner A mines a block at height r, it will not be of use to other miners like B, because miner B already has a bock at height r and one at height $r + 1$.

nodes to which he is connected.[23] Miner A continues to mine upon the block at height $r-1$, hoping to successfully mine a block at height r. He does in fact get lucky, and he mines a block at height r—he gets connected to the internet again, which allows him to broadcast his block r to other nodes. However, when miner B receives this block, it is not of much use to him/her: miner A's block is only valid on top of the block at height $r - 1$. Miner B cannot simply add the block he received from miner A on to the end of his chain (on top of the block at height $r + 1$), because mined blocks are only valid for exactly one previous block in the network—in this case, that previous block is the one at height $r - 1$. Now, to be clear: forks in the blockchain are common, and technically any fork can become the new "main" chain. However, because miner B (and most of the other nodes in the network who didn't lose internet or otherwise lose connection to the P2P network) is already 1 block ahead of the block at height r he received, he knows that the block he received from miner A is very unlikely to end up on the main chain.[24] Thus, miner B disregards the block he received, and continues to try to mine on top of the block at height $r + 1$ in hopes of being

[23]It's not particularly realistic that a miner would fail to notice that he/she lost connection to the internet. Nonetheless, we make this assumption to illustrate an important point about the blockchain.

[24]If we think back to the Poisson process discussion in the previous section, we realize that if a fork branch is a few blocks ahead of another fork branch, the probability that new blocks are mined consecutively on the shorter branch such that it outpaces the longer branch is extremely low. In fact, for every block that one branch moves ahead of another by, it becomes exponentially more unlikely that the shorter branch will ever catch up. Satoshi Nakamoto described this unique trait of blockchain in his original Bitcoin white paper [8].

the miner who finds a valid block at height $r + 2$.

2.4.2 Blockchain as a "Distributed Consensus"

The blockchain acts as the "distributed consensus" in the network, in the sense that miners who mine upon the most recent block in the their copy of the blockchain are implicitly accepting the blockchain they have as valid [1]. Sure, a miner could start mining consistently on a random fork off the main chain, but if other miners don't accept that fork on their own copies of the blockchain, the blocks the miner mined on that fork become worthless. Why is this the case? This is the case because miners only receive block rewards and transaction fees for blocks they mine *if and only if* those block ends up on the main chain [1]. Thus, the incentive of bitcoin mining rewards forces miners to implicitly reach a consensus about the state of the blockchain by mining upon the longest chain, the main chain.[25] This is one of the amazing aspects of bitcoin: because of the naturally-occurring distributed consensus mechanisms described herein, the network can afford to lack a centralized overseer. Ultimately, the technology and design of the system make a centralized body or rule enforcer unnecessary.

Let's discuss a situation in which a malicious miner wanted to alter the blockchain—it turns out that such a move would be essentially impossible because of how the blockchain, as a data structure, is designed. Each block in the chain encodes all of the information about the previous block, meaning that changing information in one block subsequently changes information it the next block (and this domino effect continues) [1]. That means that if someone tried to alter, say, a previous transaction in a block at height r, the hash of that block would change. Because the hash of block r would change, the header of the block $r + 1$ would then not correspond with the altered hash of block r. Because Bitcoin nodes in the P2P network have the ability to check the validity of a blockchain, the software would immediately detect the inconsistency between the block r's hash and the header of the next block, block $r + 1$—thus, the software would deem block r, and thus the entire altered chain of blocks, invalid. Nonetheless, even if someone could tamper with the content

[25]This paper will look at the situation in which miners try to go around this process and can actually be profitable doing so.

of the blockchain, because most nodes have their own copy of the blockchain, nodes would be able to tell that the altered chain is different from the one stored on their node. Thus, the integrity of the blockchain data structure is, by design, very reliable.

2.4.3 Forks

There are different types of forks in the blockchain, like soft forks and hard forks, but there is one type particularly relevant to this paper [1; 6; 5; 3]. The fork in question is called a bifurcation (or in rare cases, a trifurcation or a quadfurcation), and it occurs when two blocks at the same height are broadcasted in the network almost simultaneously. In this situation, some nodes will receive one of the blocks first; others will receive the other first. Bitcoin protocol suggests that, when receiving two blocks at the same height, miners are supposed to append their blockchain with whichever block reaches them first (and thus start to attempt to mine the next block on that received block) [3; 6]. When a bifurcation occurs, it is temporarily unclear which of the two blocks will become part of the main chain. When a miner in the network mines and broadcasts a block on top of one of the two blocks in the bifurcation, other miners will receive it and thus mine on top of it. This means that one of the two forks in the bifurcation is now extended, making that extended fork likely to beat out its counterpart in the long run [3; 6]. However, it should be noted that a valid block is never out of the running for being in the main chain–however, for every block mined on top of one prong in the fork, it becomes exponentially less likely for the other prong to "catch up" and overtake the fork that is longer than it. This fact relates directly to Equation 4, because the chance that a shorter chain will surpass the main chain becomes exponentially less likely with every block added to the main chain, due to the Poisson nature of block discovery.

2.4.4 51% Attacks

In our final section about blockchain, we briefly discuss something called a "51% attack" [1] [13]. Earlier we mentioned that the distributed consensus aspect of the blockchain means that a miner has little incentive to mine on a fork other than the main chain, because his/her blocks on the fork are unlikely to be accepted by other miners. Thus, given that miners want to receive the rewards for the blocks they mine, they will implicitly agree with the rest of the network by simply mining on the main chain. This changes, though, if a group in the network were to posses over 50% of the hashing power in the network [1; 13]. If this were the case, such a mining group would be able to turn essentially any fork into the main chain because it could reject blocks mined by the other 49% or fewer miners in the network and fully control the blockchain. This concept is well described by a academic paper about bitcoin: "Once a miner pool...reaches a majority, it controls the blockchain...a majority pool can collect all the system's revenue by following the prescribed Bitcoin protocol, and ignore blocks generated outside the pool...At this point, the currency is not a decentralized currency as originally envisioned." [3; 4] This situation is theoretical and has not happened in the bitcoin system. However, it is an important concept to understand, because it shows that the distributed consensus and decentralized mechanisms that make bitcoin unique breakdown if any one participant gains more than 50% of the hashing power in the network.

3 Model

3.1 Motivation

As we described in the previous section, the process of mining, on a high, non-technical level, is reasonably simple. A miner collects transactions into a "block," hashes the header of that block with different random numbers (nonces) into a SHA-256 hashing function, and hopes to generate an output from the hashing function that is below a certain threshold. When and if a miner discovers a nonce that successfully hashes a value below that threshold, the miner immediately broadcasts that "block" with the the proper proof-of-work to the rest of the network. The above process is what the Bitcoin protocol, the agreed-upon "rules" that miners are supposed to follow, instructs miners to do [1; 8]. Before "Selfish Mining" was discovered as a viable strategy, it was thought that the above protocol-following mining process was incentive compatible—in other words, there was thought to be no incentive to "cheat" the system. As it turns out, however, selfish mining can potentially increase expected mining revenues.

A substantial amount of literature has assessed the feasibility and profitability of engaging in subversive mining behavior—that is, scholars who study the behavior of cryptocurrency blockchains have built varying models on how one can subvert the "protocol" of a given blockchain and the results of such actions [3; 6; 5; 4]. In particular, there is a model called "Selfish Mining" that defines a decision process that a miner can follow to increase his/her relative revenue (selfish miner revenue as a fraction all revenue earned in the system) [3]. Once a model of this behavior was released in a 2013 paper, other scholars complicated the model by accounting for other variables in the system (i.e. block propagation rate, block size, etc.) while some have tried to further optimize the selfish mining strategy [4; 3; 5]. This paper aims to de-generalize a key assumption under which essentially all of these models operate: the current models focus on the effects of one selfish miner in the network, whereas my paper focuses on the effects of many selfish miners in the network.

In addition to mapping out the decision process for the selfish mining strategy via an algorithm, the 2013 paper also uses a Markov chain model and its state probabilities to determine the expected relative revenues of miners following this strategy [3]. In this paper, I aim to see if those theoretical expected revenues hold up in the case that there are many selfish miners acting individually—using a bitcoin network simulator, we will be able to directly compare revenues of network participants to those that the theoretical model would predict.

In order to go about this investigation, we have to start by clearly defining the selfish mining strategy as it is traditionally modeled (and understand exactly how it differs from the staus quo, protocol-based mining process). We will then go through the calculations of expected revenues from this strategy using a Markov chain model and its state probabilities.[26] Finally, we will perform simulations of the bitcoin network to see how true mining revenues relate to those that the Markov-chain-based revenue model would predict.

Discrete-Time Markov chains allow us to calculate theoretical revenues associated with different mining strategies, because the blockchain takes on different states with dif-

[26]This Markov chain model, as mentioned earlier, was built by the authors of [1] and is also used widely by other authors who discuss the selfish mining strategy.

ferent probabilities due to the Poisson nature of block creation.[27] While mining activities and blockchain changes occur in continuous time, what really matters to a mining strategy are the discrete events of block creation—thus, using a Markov chain that disregards the continuous time nature of bitcoin mining is not an issue. In other words, in the context of calculating miner revenues, the time in between mined blocks is much less important than who mines each block and what he/she does with that block. In the first section of model building, we will use a Markov chain to calculate theoretical revenues for a miner who follows Bitcoin mining protocol, which we will call an "honest" miner.

3.2 Honest Mining and Expected Revenue Model

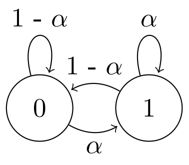
Before discussing the details of the selfish mining strategy decision process and subsequently the expected revenues associated with that strategy, we must clearly review the honest mining process and its associated expected revenues as a base for comparison. We will use a Markov chain to convert what we already know about the protocol-following "honest" miner into expected revenues for that miner.

Say we have an honest miner who possesses the fraction α of the hashing power in the network. All other miners in the network combined make up the rest of the $1-\alpha$ fraction of hashing power. In Figure 5, let state 1 be the case in which this honest miner has just successfully mined a block and starts the mining process all over again in hopes of finding the next block.[28] From state 1, the honest miner has probability α of mining the next block, which corresponds with staying in state 1 again. This also means that the other miners in the network have probability $1 - \alpha$ of mining the next block (instead of the honest miner mining the next block), which corresponds with the honest miner transitioning from state 1 to state 0. Now in state 0, the honest miner has probability α of mining the next block and thus moving to state 1, etc. As we can see, the pattern is repetitive and simple: every time a block is mined in the network, if it's the honest miner in question who mined it, he/she moves to state 1 from whichever state he/she was previously in; if it's any other miner who

[27]Discussed in detail in Section 2.3.3.

[28]It is important to remember that our term of "honest" miner simply describes the mining process from Section 2 of this paper. It is not new—we just give it a name to have a counterpart to "selfish" miner.

Figure 5: This simple Markov chain can be used to formally define expected theoretical (relative) revenues associated with honest mining. As defined by general Markov chain drawing conventions, the 1 above an arrow is not required, but we use it for emphasis of the guaranteed transition from state 1 to state 0.



mined it, the honest miner in question moves to state 0 from whichever state he/she was previously in.

We can use this Markov chain to solve for relative revenue of an honest miner, by finding the state probabilities associated with this Markov chain and then assigning revenues to those states. The ratio of time spent in either states combined with the revenues gained in those states is a simple yet powerful way to understand overall revenue generation from a stochastic process like this one. The results of these calculations will serve as the base relative revenue to which we can later compare the expected revenues of selfish miners. We can solve for the steady state of the Markov chain, where our steady state vector, $\mathbf{s}$, satisfies $\mathbf{s}Q = \mathbf{s}$, where Q is the transition matrix associated with the Markov chain in Figure 5.

$$\mathbf{s}Q = \begin{bmatrix} s_0 & s_1 \end{bmatrix} \begin{bmatrix} 1-\alpha & \alpha \\ 1-\alpha & \alpha \end{bmatrix} = \begin{bmatrix} s_0 & s_1 \end{bmatrix} \tag{8}$$

In the above matrix equation, we want to solve for the components of $\mathbf{s}$, which give us the long-term fraction of time that will be spent in each state. We have,

$$s_0(1-\alpha) + s_1(1-\alpha) = s_0$$

$$s_0\alpha + s_1\alpha = s_1$$

And by definition of a steady state vector, all entries add up to 1, so we have

$$s_0 + s_1 = 1$$

Substituting $s_0 = 1 - s_1$ into the equations above, we solve to to get

$$\begin{aligned} s_0 &= 1 - \alpha \\ s_1 &= \alpha \end{aligned} \tag{9}$$

And finally to calculate expected relative revenue, we multiply the revenue (in terms of number of blocks awarded in each state) associated with each state by the time spent in that state. Thus, we arrive at:

$$E(r_{h,rel,model,\alpha}) = 0 \cdot s_0 + 1 \cdot s_1 = 1 \cdot \alpha = \alpha \tag{10}$$

This solution confirms a strikingly simple tenet of bitcoin mining: the number of blocks a miner finds as a fraction of all blocks discovered in the network is equal to that miner's fraction of the network's hashing power (in this case, α) [1; 8; 3; 6].[29]

Let's briefly make clear the meaning of each subscript in Equation 10: h means that this metric relates to honest miners (rather than selfish ones), *rel* denotes relative revenue, *model* signifies that this result is from our mathematical model (not a simulation), and α denotes a hashing power fraction.

We can now move on to discussing the selfish mining strategy. After understanding the strategy, we will be able to perform a similar, but much more complex, expected revenue calculation using a Markov chain model once again.

3.3 Selfish Mining Strategy

The selfish mining strategy, laid out by Eyal and Sirer in their 2013 paper, contradicted the strongly-held belief that bitcoin mining protocols were fully incentive compatible [3]. While I will approach my description and walk-through of the strategy slightly differently than the aforementioned authors due, it is important to note that those authors deserve full credit

[29]Stale blocks, blocks that do not end up on the main chain, do not award miners revenue as we have discussed. So, technically, a miner's fraction of revenue is the number of blocks they have mined on the main chain as a fraction of all blocks mined on the main chain. This metric, called "relative revenue" is the one upon which we base our results throughout the rest of this paper.

for the formulation of the selfish mining model [3]. We will also present the coded algorithm that those authors use to explicitly define the steps of the strategy [3].

The core concept of the selfish mining strategy is that selfish miners attempt to “waste” the hashing power of honest miners by causing those honest miners to create blocks that will not end up on the main blockchain. As a result of this wasted hashing power, fewer blocks will end up on the blockchain, meaning that difficulty will decrease (see Section 2.3.3), leading to higher revenues for selfish miners. A key fact to remember in understanding this strategy is that only validated blocks that end up on the main chain will “reward” their miners with the block reward and transaction fees. There is a name for blocks that are valid and successfully mined but don’t end up on the main blockchain—these are called ”stale blocks” [4; 5; 3]. Every second a miner spends trying to hash a block and nonce below the target associated with a block that is destined to become “stale,” the lower their expected revenue will be. So, the aim of a selfish miner is to cause as many of its opponents’ (honest miners’) blocks to become stale, all the while not compromising his own expected number of blocks mined on the main chain. If a selfish miner can achieve this subversion in some way, he/she has the chance to increase his/her long-term expected relative revenue: over time his/her strategy (in some cases, as we will see) will lead to the same number of blocks in the main chain all the while causing decreased difficulty due to wasted hashing power of honest miners.

We will imagine there exists some miner (or group of miners working together) who engages in the selfish mining strategy. For sake of later numerical analysis, we say that this miner or group of miners has/have α fraction of the hashing power in the network. This implies that everyone else in the network, which we will assume to be all mining honestly and abiding by protocol, make up the rest of the $1 - \alpha$ fraction of hashing power in the network.

In order to understand the selfish mining strategy, we need to walk through different mining events to see what actions the selfish miner would take. We will discuss these mining events in the following sections.

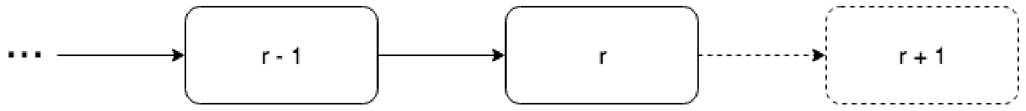


Figure 6: Status-quo of the main chain

3.3.1 Another Miner Mines a Block

First, we begin with the most common case of the strategy: a selfish miner will try to mine a block on top of the most up-to-date main chain. This approach is no different than that of an honest miner—the selfish miner needs to successfully mine a block before he/she can even start to engage in the "selfish" part of the strategy. Representing this simple situation graphically in Figure 6, we show the newest portion of the blockhain, with the r^{th} block representing the most recently published mined block on the main chain, which is at height r. The dotted-box block at height $r + 1$ is theoretical—it hasn't been found yet, but eventually it will be mined by some miner in the network. When that block is mined, the process starts all over again, and miners will be mining upon the block at height $r + 1$ in hopes of generating the a block at height $r + 2$. So long as honest miners continue to mine and broadcast their mined blocks, the chain in Figure 6 will grow horizontally as shown. However, if a selfish miner mines a block, the situation will change—we now use the next section to analyze this case.

3.3.2 The Selfish Miner Mines a Block

Now we consider the case in which the selfish miner mines a block. Let's say that the selfish miner mines a new block upon the block at height r, but the miner keeps it to him/herself. We will call this the $(r + 1)'^{\,th}$ (with the intentional $'$ symbol) block, because it has the potential to be at height $r + 1$ (on the main chain, by definition of height), but we do not yet know if that will be the case. Another equivalent way to describe this is that the block is a valid next block after the block on the main chain at height r, but it has yet to be broadcasted to any nodes in the network, and thus cannot yet be part of the main chain. Therefore, the selfish miner now has a "private" chain of 1 block. We update the diagram to account for this change—refer to Figure 7.

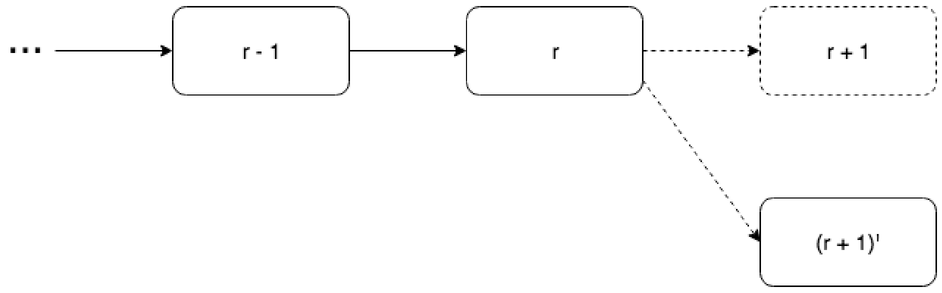


Figure 7: The selfish miner has now mined a block that is even with height $r+1$ in the main chain, but the block's fate is not yet determined.

Note that solid-bordered boxes represent blocks that have been mined successfully, and solid-line arrows represent connections between blocks on the main chain. Dotted-border boxes are blocks that haven't been mined yet but will be by someone eventually, and dotted arrows are connections between blocks that have not yet materialized on the main chain, but may be eventually. Because the $(r+1)'\ ^{th}$ block is successfully mined but not on the main chain, it has a solid-border but a dotted leading arrow.

Now at the $(r+1)'\ ^{th}$ block, the selfish miner will take the following action: he/she will immediately start to mine on top of the $(r+1)'\ ^{th}$ block, hoping to extend his private chain to the $(r+2)'\ ^{th}$ block.

3.3.3 Another Miner Mines a Block, Initiates Bifurcation Race

Let's analyze the case in which the public discovers a block at height $r+1$ before the selfish miner can mine block $(r+2)'$. An depiction of this situation is shown in Figure 8 in which another miner in the network mined and broadcasted a block at height $(r+1)$. At this point, the selfish miner will immediately broadcast his own block, creating a bifurcation in the main chain: there are now two valid blocks on the main chain at height r. Figure 9 depicts this scenario.

When this happens, the protocol suggests that miners should mine upon whichever of the two blocks arrives at their node first, as discussed earlier. As these two blocks at height $r+1$ are mined upon, only one will prevail—once one of these two blocks is extended, the new official main chain will contain whichever of the two blocks ($r+1$ or $(r+1)'$) is

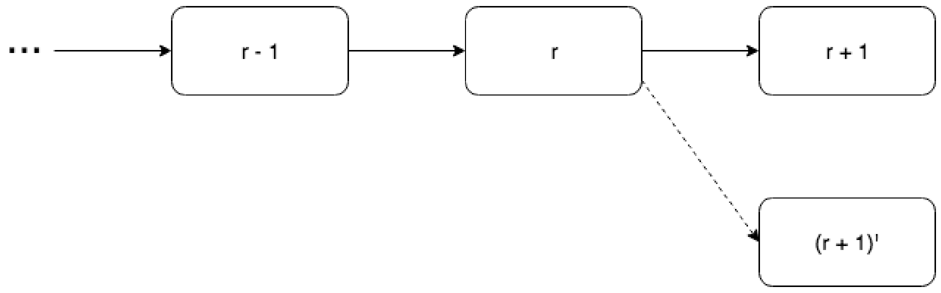


Figure 8: Some other miner in the network has mined and broadcasted a block at height $r + 1$ while the selfish miner has a private chain of length 1 at equivalent height

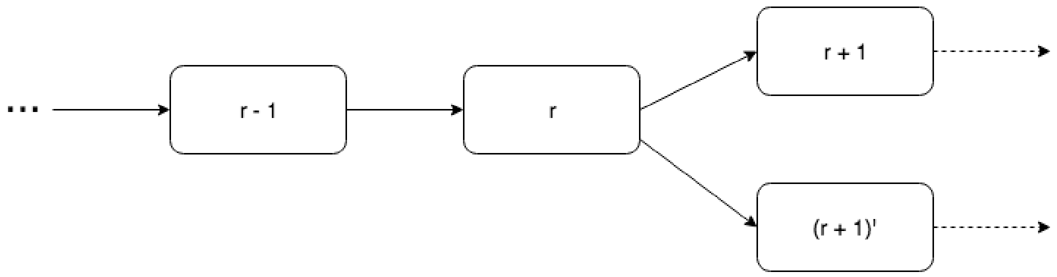


Figure 9: The selfish miner has now broadcasted his private chain of length 1, creating a bifurcation in the main chain at height $r + 1$. A "toss-up" or "bifurcation race" will now begin as a result of this bifurcation.

extended first, turning one of the two dotted lines into a solid line in the Figure 9. If the selfish miner's block is the one that gets extended, he/she is in luck: the block that he/she originally withheld from the public is now included in the main chain, meaning he receives the block reward and transaction fees from his block (the $(r + 1)^{th}$ block). While this outcome has the potential to be successful for the selfish miner, there is a chance that he loses out: if the $r + 1$ block gets extended before the $(r + 1)'^{th}$ block, the selfish miner has likely forever lost the revenue for his mined block that he would have received had he been mining honestly.

3.3.4 The Selfish Miner Extends the Lead

Now let's look at the case in which the selfish miner actually extends his/her private chain to $(r + 2)'$ before the state of the main chain has even gotten to $r + 1$. This situation is depicted in Figure 10.

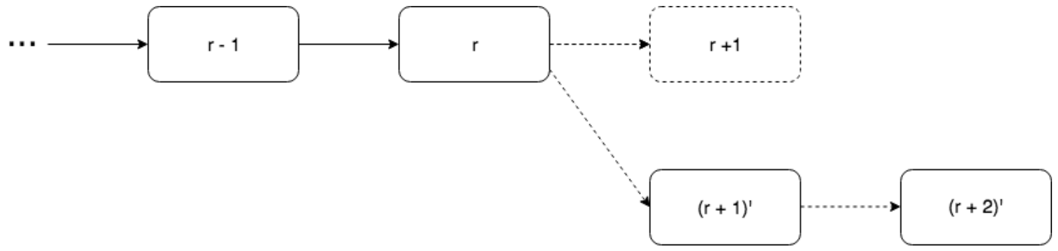


Figure 10: The selfish miner has now extended his/her private chain two blocks past the current height of the main chain.

As could be guessed, the selfish miner will continue to mine on block $(r+2)'$ in hopes of extending to a 3-block lead at the $(r+3)'^{th}$ block. This process will stop as soon as a block is mined on the main chain that reduces the lead of the the selfish miner's private chain over the main chain to 1, at which point the selfish miner will publish all of the blocks he has on his private chain. Let's show one example of this in Figure 11, in which the miner sees his/her lead drop to 1 block and publishes his/her private chain of 2 blocks: this private chain will get adopted onto the main chain, making the $r+1$ block stale.

This will lead the public to adopt his newly-published branch, because it is longer, by 1 block, than the previously-main chain. This is the case in which the selfish miner has achieved his/her goal: he/she received the revenue associated with the blocks he/she mined (as he would have if he/she were following the rules as an honest miner) but he/she also hurt the profit of the other miners in the network: they all wasted time mining blocks after the r^{th} block that ended up getting replaced / overrode by the longer private chain that the selfish miner broadcasted. Thus, the selfish miner's strategy of withholding blocks until a strategically-planned release time leads other miners to waste their hashing power on blocks that will not end up rewarding revenue, which increases the selfish miner's relative revenue.[30]

Now that we have laid out the essential logic of the selfish mining strategy, we will show the algorithm of the selfish mining strategy, as defined by Eyal and Sirer in their 2013 paper, to formalize and summarize our description of the strategy thus far [3]. In fact, this

[30]Additionally, the decreased difficulty that will arise due to the production of more stale blocks than normal helps the selfish miner's relative revenue.

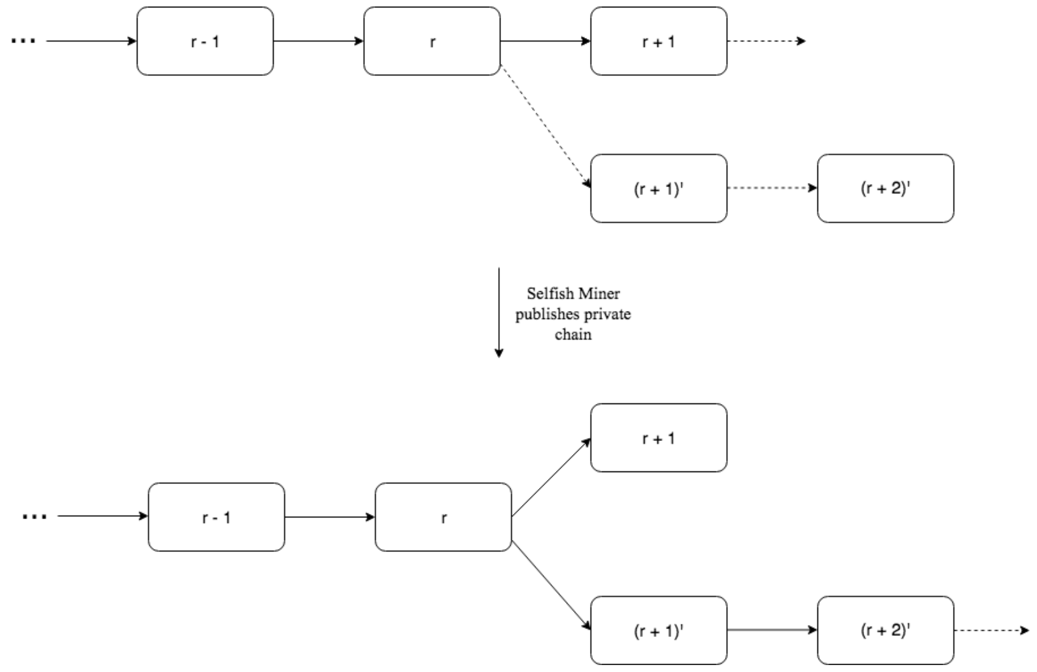


Figure 11: The selfish miner's multi-block private chain lead over the public main chain has now been reduced to a lead of 1 block. When this happens, the selfish miner broadcasts his/her private chain which replaces the block at height $r + 1$, and thus because the selfish miner's broadcasted private chain extends upon the r^{th} block to a greater height than $r + 1$, the private chain becomes part of the main chain.

algorithm will be used in our simulation later as a way to simulate the actions of selfish miners in the bitcoin network. Figure 12 shows the algorithm.

3.4 Selfish Mining and Expected Revenue Model

In our Markov chain model for the selfish miner, we define each state as the number of blocks the private chain is ahead of the public chain (with one exception to be discussed shortly). Thus, we expect to see states $0, 1, 2, ...$ in the model. We must introduce an additional state, however, to account for the bifurcation race that we discussed earlier. As Eyal and Sirer do, along with other authors, we can call this state $0'$ [3; 4; 5]. In the case of a bifurcation race, the number 0 makes intuitive sense because a bifurcation in the blockchain means that the previously private chain is at the same height of the main chain (making the length differential between the main and private chain 0, hence state 0). The prime symbol ($'$)

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.
```

Figure 12: This algorithm, which appears in the 2013 paper by Eyal and Sirer, formalizes the selfish mining strategy that we have discussed [3]

differentiates state $0'$ from normal state 0 because state 0 accounts for cases in which the blockchain only has one main chain and miners are simply mining on most recent block in the main chain. Now we address the one exception mentioned earlier: technically when the chain is in state 2 (meaning the private chain is 2 blocks ahead of the main chain) and a miner in the network mines a block, the differential between the main chain and private chain is briefly 1. This would technically imply the Markov chain transitions to state 1 briefly, but we do not model this nuance, because the case of a public block discovery when in state 2 always results in the selfish miner broadcasting his/her private chain.[31]

Now armed with an understanding from the previous section of how the main chain and private chain can diverge under the selfish mining strategy, we will use the Markov chain to in Figure 13 to mirror all of the different possible states and transitions. In state $0, 1, 2....,$

[31]And thus a return to the status quo of the blockchain (state 0).

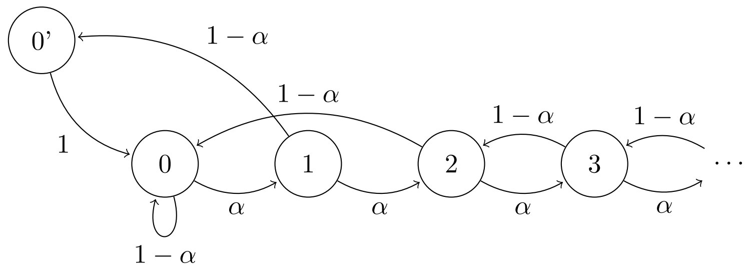


Figure 13: This Markov chain represents the number of blocks that the selfish miner's private chain is ahead of the main chain, with state $0'$ representing the unique case of a bifurcation race.

the selfish miner has probability α of extending his/her private chain by one block, because we know that one's probability of discovering a block is equivalent to his/her fraction of hashing power in the network.

When a selfish miner finds a block he/she will never move in one step back to state 0. Rather, the selfish miner will stay in state 1, meaning he/she has successfully mined a block not yet broadcasted to other nodes, and will try to continue to build upon his/her "private" chain of length 1. If another miner in the network discovers a block, that other miner will broadcast his/her block: this will cause the selfish miner to immediately broadcast his 1-block private chain, thus initiating a bifurcation race between two blocks. As we know, this situation is represented by state $0'$. In the case that chain reaches state 2, however, all subsequent states, representing further leads of the selfish miner's private chain, will be transitioned to with probability α. Any state transition that occurs when someone other than the selfish miner mining a new block has probability $1 - \alpha$.

3.4.1 Steady State Probabilities

Using the Markov chain in Figure 13, we can calculate its steady state probabilities—this will eventually allow us to calculate the expected revenue of the selfish mining strategy by combining the steady state probabilities with expected revenue at each state.

Because the Markov chain has an infinite tail, we have to be careful when calculating steady state probabilities. Because the pattern for all states 3,4,... are identical, it will make steady state calculation of the tail states feasible. To find the steady state probabilities, we use the same method used in the honest mining expected revenue calculations. We use the fact that $\mathbf{s}Q = \mathbf{s}$ for a steady state vector, $\mathbf{s}$, and a transition matrix, Q, which encodes the transition probabilities of the Markov chain.[32]

$$\mathbf{s}Q = \begin{bmatrix} s_{0'} & s_0 & s_1 & s_2 & s_3 & . & . \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & . \\ 0 & 1 \text{ - } \alpha & \alpha & 0 & 0 & . \\ 1 \text{ - } \alpha & 0 & 0 & \alpha & 0 & . \\ 0 & 1 \text{ - } \alpha & 0 & 0 & \alpha & . \\ 0 & 0 & 0 & 1 \text{ - } \alpha & 0 & . \\ . & . & . & . & . & . \end{bmatrix} =$$

$$\begin{bmatrix} s_{0'} & s_0 & s_1 & s_2 & s_3 & . & . \end{bmatrix}$$

Multiplying out the $\mathbf{s}Q$ to generate a system of equations, we get the following

$$\begin{gathered} (1 \text{ - } \alpha)s_1 = s_{0'} \\ s_{0'} + (1-\alpha)s_0 + (1-\alpha)s_2 = s_0 \\ \alpha s_0 = s_1 \\ \forall i \geq 2 : \alpha s_i = (1-\alpha)s_{i+1} \\ s_{0'} + \sum_{i=0}^{\infty} s_i = 1 \end{gathered} \tag{11}$$

Where the final equation is true because all steady state vectors sum to 1 by definition. Using the equations in (11) above, we substitute the first and third into the second such that that the second will be in terms of s_1 and s_2. The substitutions lead to

$$(1-\alpha)s_1 + (\tfrac{1-\alpha}{\alpha})s_1 + (1-\alpha)s_2 = \tfrac{1}{\alpha}s_1$$

Solving for s_2 in terms of s_1, we have

$$s_2 = (\tfrac{\alpha}{1-\alpha})s_1$$

[32]In the equations, dots indicate that the steady state vector and the transition matrix technically extend infinitely.

Using this expression for s_2, we plug into the the fourth equation in (11) which allows us to spot a pattern for expressing any s_i with $i \geq 2$ in terms of s_1. So we have,

$$s_i = (\tfrac{\alpha}{1-\alpha})^{i-1} s_1$$

Now, we make use of the fifth equation in (11). The right side of the equation is 1, and on the left side, we can pull out all the s_i that can be replaced with expressions in terms of s_1. So we write the fifth equation in (11) as

$$s_{0'} + s_0 + \textstyle\sum_{i=1}^{\infty} s_i = 1$$

and we can plug in for s_0, s_1, and for the general s_i in the sum.

$$s_{0'} + s_0 + \textstyle\sum_{i=1}^{\infty} s_i = 1$$

and then we get

$$(1-\alpha)s_1 + \tfrac{1}{\alpha}s_1 + \textstyle\sum_{i=2}^{\infty}[(\tfrac{\alpha}{1-\alpha})^{i-1} s_i] = 1$$

Solving for s_1 (see Appendix for details), we get the following

$$s_1 = \tfrac{\alpha - 2\alpha^2}{2\alpha^3 - 4\alpha^2 + 1}$$

Finally, plugging this expression for s_1 into equations first, second, and third equations of (11), we obtain the rest of the steady state probabilities. We list them all together below:

$$\begin{aligned} s_{0'} &= \frac{\alpha - 2\alpha^2}{\alpha(2\alpha^3 - 4\alpha^2 + 1)} \\ s_0 &= \frac{(1-\alpha)(\alpha - 2\alpha^2)}{1 - 4\alpha^2 + 2\alpha^3} \\ s_1 &= \frac{\alpha - 2\alpha^2}{2\alpha^3 - 4\alpha^2 + 1} \\ \forall i \geq 2 : s_i &= \left(\frac{\alpha}{1-\alpha}\right)^{i-1} \frac{\alpha - 2\alpha^2}{\alpha(2\alpha^3 - 4\alpha^2 + 1)} \end{aligned} \tag{12}$$

3.4.2 The Calculation of Expected Revenue

Calculating expected revenue using the state probabilities was a relatively trivial task for the case of the honest miner. For the selfish miner, however, we have to consider both the state probabilities and the route via which a selfish miner arrives to a given state. For example, moving to state 0 from state $0'$ and state 2 award different revenues. Thus, we need to go through all of the different cases involving movement among the markov chain states in order to calculate the expected revenue associated with engaging in the selfish mining strategy. Before going through the different cases, though, we need to add one more variable to the model that is important in determining revenue. In the case of the bifurcation race, we discussed the fact that miners will mine on whichever of the two blocks in the fork (see Figure 9 as a reminder of this case). Instead of working under the assumption that $\sim 50\%$ of miners mine on one branch of the fork, and the other $\sim 50\%$ mine on the other, we need to define a variable for the fraction of miners who choose to mine on one prong of the fork versus the other. This is important, because a lot of factors (such as network latency, to be discussed in our simulation section at length, the exact time a block was broadcasted, etc.) can make the ratio dramatically different than 50%. So, we let ϕ be the fraction of miners who choose to mine on the block released by the selfish miner, while $1 - \phi$ is the fraction of miners who decide to mine on the other block in the bifurcation race.

We can now define a set of cases, each associated with a different revenue amount (i.e. number of blocks mined on the main chain), and then match each case's associated revenue with its corresponding state probability and transition frequency to that state (given by the Markov chain and steady states in Figure 13 and the Equations in (12), respectively).

3.4.3 Cases and Their Revenues

In each case, we will discuss the qualitative nature of the scenario, then calculate the contribution of that scenario to the expected revenue of a selfish miner and that of the honest miners. We will organize the cases into three general categories: cases that involve a bifurcation race, cases that involve selfish miners having a private chain that leads the main chain by two or more blocks, and cases that do not fit into the aforementioned two categories. To avoid convoluted terminology, we will use the letter S to refer to the selfish mining group, and H to refer generally to the honest miners in the network. We also refer to height r as an arbitrary height in the blockchain.

1. **Bifurcation of the Main Chain**
   (a) A bifurcation race has just commenced, and the result of the race is not yet determined.
      - This case represents the start of a bifurcation, meaning that the revenue of S or H cannot yet be determined. It will soon be determined by one of the following cases, (b)-(d).

   (b) In a bifurcation race between a block from H and a block from S, S's block wins. The block that extended S's block causing them to win happened to be mined by H.
      - S gains a revenue 1 because its block won in the bifurcation race. H also receives a revenue of 1 block because it mined on top of S's block.

   (c) In a bifurcation race between a block from H and a block from S, S's block wins. The block that extended S's block causing them to win happened to be mined by S.
      - S gains a revenue 2 blocks because its block won in the bifurcation race and it mined the next block as well. H gains a revenue of 0 blocks because it lost the bifurcation race and did not mine the subsequent block.

   (d) In a bifurcation race between a block from H and a block from S, H's block wins. The block that extended H's block causing them to win happened to be mined by H.
      - H gains a revenue 2 blocks because its block won in the bifurcation race and it mined the next block as well. S gains a revenue of 0 blocks because it lost the bifurcation race and did not mine the subsequent block.

2. **Selfish Miner Leads by 2 or More Blocks**

(a) H mines a block on the main chain, which reduces the lead of S's private chain over the main chain to only 1 block. This causes S to immediately reveal its private chain of 2 blocks.

- H gains a revenue of 0 blocks because its chain gets overridden by the chain that S releases, thus causing H's blocks to no longer be part of the main chain meaning they do not reward any revenue. S gains a revenue of 2 blocks because it broadcasted its 2 block private chain, making that now part of the main chain.

(b) H mines a block on the main chain, so S releases a block from its private chain. Thus, the lead of S's private chain over the main chain is reduced by 1 block. However, S's private chain still leads the main chain by at least 2 blocks.

- H gains a revenue of 0 blocks, because its temporary 1-block lead gets tied by the 1 block that S releases. S still has blocks on its private chain, which it will release later. Nonetheless, S gains a revenue of 1 block for now because its 1 released block in this case is destined to stay on the main chain due to the fact that S has more blocks for later release.[33]

3. **Other Cases**

(a) S does not have a private branch, and H mines a block on the main chain.[34]

- H gains a revenue of 1 block, and S gains a revenue of 0 blocks.

(b) S mines a block and thus extends its private chain by 1 block.

- H gains a revenue 0 because it did not mine a block. S also gains a revenue of 0 blocks: even though S did just mine a block, the fate of the that block is not yet determined. It could end up on the main chain, but it may not. Thus, in this case, the revenue of S is 0 blocks.

[33]Having more blocks "stocked up" on its private chain means that S can stop H from getting ahead of S in subsequent block mines. Thus, we consider the 1 block received by S to have a secure spot on the main chain for the long term.

[34]Note that this is the status quo case. All miners in the network, H and S, are hoping to mine block, and S has no private chain.

3.4.4 Final Expected Revenue Results

We now have all of the information needed to build an expression for the expected revenue of the selfish miner and the revenue of everyone else in the network (the honest miners), in terms of α and ϕ. The expected revenue equations for both the selfish miners and the rest of the network will consist of a weighted average of revenue associated with the cases above and the probabilities of those cases (using the steady state probabilities found in Equations (12)).

Let $r_{self,abs,model,\alpha}$ and $r_{h,abs,model,1-\alpha}$ be the number of blocks found by the selfish miner and the rest of the network, respectively. Once again, we keep our subscript notation consistent with Equation 10; in this case, *abs* indicates that these revenues are absolute, not relative. Absolute means that we are simply counting the number of blocks mined in the main chain, rather than dividing by the total number of blocks in the main chain as we would to calculate relative revenue. Using the cases above, the Markov chain model, and the steady state probabilities, we can generate both $E(r_{self,abs,model,\alpha})$ and $E(r_{h,abs,model,1-\alpha})$. As shown in (13), we use the braces to denote which of the cases from Section 3.4.3 generates that portion of the equation. All of the scenarios that generate revenue of 0 do not contribute to the expectations of revenues, of course. That is the reason why not all of the cases from Section 3.4.3 are referenced in the equations.

$$\begin{aligned} E(r_{self,abs,model,\alpha}) &= \overbrace{2 \cdot s_{0'}\alpha}^{1(c)} + \overbrace{1 \cdot s_{0'}\phi(1-\alpha)}^{1(b)} + \overbrace{2 \cdot s_2(1-\alpha)}^{2(a)} + \overbrace{1 \cdot P(i \geq 2)(1-\alpha)}^{2(b)} \\ E(r_{h,abs,model,1-\alpha}) &= \underbrace{1 \cdot s_{0'}\phi(1-\alpha)}_{1(b)} + \underbrace{2 \cdot s_{0'}(1-\phi)(1-\alpha)}_{1(d)} + \underbrace{1 \cdot s_0(1-\alpha)}_{3(a)} \end{aligned} \tag{13}$$

So, if we let $r_{self,rel,model,\alpha}$ be the relative revenue of the selfish miner, the equation for the expectation of that variable is as follows:

$$E(r_{self,rel,model,\alpha}) = E\left(\frac{r_{self,abs,model,\alpha}}{r_{self,abs,model,\alpha} + r_{h,abs,model,1-\alpha}}\right)$$
$$= \frac{E(r_{self,abs,model,\alpha})}{E(r_{self,abs,model,\alpha}) + E(r_{h,abs,model,1-\alpha})}$$

by linearity of expectation.

Finally, plugging in the expectations which we found earlier, we have: [35]

$$E(r_{self,rel,model,\alpha}) = \frac{\alpha(4\alpha + \phi - 2\alpha\phi)(1-\alpha)^2 - \alpha^3}{1 - \alpha(1 - \alpha^2 + 2\alpha)} \tag{14}$$

Where we use subscript notation corresponding with that of Equation 10 because this (Equation 14) is the selfish-mining relative revenue counterpart to the honest-mining relative revenue. Thus, in Equation 14, *self* denotes that this is relative revenue is associated with a selfish miner, *rel* denotes relative revenue, *model* signifies that this result is from our mathematical model (not a simulation), and α denotes a hashing power fraction of the miner in question.

3.5 Honest Mining Revenue vs. Selfish Mining Revenue

We can now chart the relative revenues associated with honest and selfish mining, for different values of α and ϕ, depicted in Figure 14. The line in the graph representing honest mining relative revenue comes from Equation 10 (found on page 32), whereas the curves associated with the selfish mining relative revenues (at different ϕ values) come from Equation 14.

Observation 1: The marginal returns to hashing power for a selfish miner (more than linear) are greater than those for an honest miner (linear). Furthermore, the expected relative revenue for a selfish miner is always higher than that of an honest miner when a

[35] We omit the copious algebra that brings us to this result, but it is simply solved for by plugging the corresponding state probabilities into the $\frac{E(r_{self,true,model,\alpha})}{E(r_{self,true,model,\alpha})+E(r_{h,abs,model,1-\alpha})}$ equation. This result matches that of Eyal and Sirer, serving as a check on the algebra. Serving as a second check, our graphs are consistent with those in Eyal and Sirer.

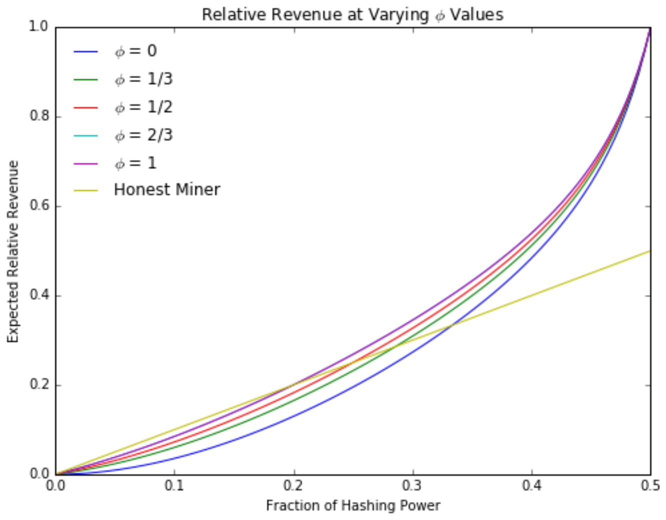


Figure 14: This plot shows the expected relative revenues of the selfish mining strategy at different values of ϕ and α, along with the expected relative revenue of honest miners solved for in Equation 10.

selfish miner possesses $\frac{1}{3}$ or more of network's total hashing power, regaurdless of ϕ.

Observation 2: The fraction of miners the mine on the selfish miner's block in the case of a bifurcation block race (ϕ) can make a significant impact on the profitability of the selfish mining strategy

Observation 3: The most important finding from all of the analysis in this section is the condition under which selfish mining awards more relative revenue than is expected from the hashing power of that miner. In terms of ϕ. we solve for the value for which $E(r_{self,rel,model,\alpha}) > E(r_{h,rel,model,1-\alpha})$, which becomes $\dfrac{\alpha(4\alpha+\phi-2\alpha\phi)(1-\alpha)^2-\alpha^3}{1-\alpha(1-\alpha^2+2\alpha)} > \alpha$. We arrive at that condition[36]:

$$\frac{1-\phi}{3-2\phi} < \alpha \tag{15}$$

[36]Eyal and Sirer arrive at this exact bound. We omit the algebra that arrives at this inequality.

where $\alpha < \frac{1}{2}$.

Now that we have shown the profitability of the selfish mining strategy in theory, how can we be sure it holds in practice? Well it turns out that these predicted relative revenues do in fact hold in practice, when the network is operating under simple conditions..[37] Specifically, Eyal and Sirer, along with other authors of papers revolved around this topic, have run Bitcoin simulations confirming that the the simulated results closely match the curves in the Figure 14 when there is one selfish miner, of hashing power α, and ϕ is artificially set at different levels [3]. However, what happens when network conditions are more nuanced and many selfish miners are participating rather than just one? My paper aims to answer that question. We seek to find out whether expected revenue results hold up in the case in which there are multiple selfish miners in the network, and ϕ cannot be explicitly controlled for. Additionally, we will see in the subsequent section why ϕ is a useful variable for simple calculations of expected revenue, but that estimating ϕ in the Bitcoin network is very complicated. Network latency, a proxy for ϕ, which we will denote as ω, turns out to be a more realistically-estimable variable in the Bitcoin network.

Now armed will a thorough intuition on how selfish miners act and what results those actions tend to warrant under simple conditions, we move to the main analysis of this paper.

4 Simulation and Results

4.1 De-Generalizing the Selfish Mining Strategy

The model described in the previous section serves as an excellent base for undertanding the effects of colluding mining groups on the Bitcoin network. However, the assumptions of said model cause it to be oversimplified compared to reality—we seek to answer whether the results of the model hold under more realistic network conditions.

The key assumption that essentially all papers analyzing selfish mining operate under is that there is a set of miners (likely as part of a mining pool) that work together to engage

[37]By simple network conditions, I refer to the fact of there being just one selfish miner in the network and a ϕ value that is controlled directly by those running the simulator.

in the selfish mining strategy—together these miners compose the fraction α of all hashing power in the network. Then, the rest of the miners, who make up the other $1 - \alpha$ fraction of hashing power, are all honest (and it is irrelevant the distribution of hashing power of miners within the honest fraction). To be clear, the colluding α fraction of the network acts totally as one miner, such that all miners that compose that fraction work to build one private chain and follow the selfish mining decision process as a team. These assumptions allow for clean analysis of the viability of the selfish mining strategy, but such a model fails to account for the more realistic possibility that there is not just one group, but rather many groups, participating in selfish mining. This portion of the paper takes on the following three questions: What are the effects on revenues of all participants in the network when many parties engage individually in the selfish mining strategy? How do these effects change when the number of selfish miners and/or their respective hashing powers are altered? Finally, how do attributes of the P2P network, such as network latency, influence these effects?

In order to test how the model generated in the previous section holds up when there are many selfish miners participating in the network is by simulating the actual Bitcoin network. With a simulator capable of mimicking the bitcoin network over some set time interval, we can record accrued revenues and other statistics to answer the questions posed in this paper. We will now discuss the details of the Bitcoin network simulator used for this analysis.

4.2 A Bitcoin Network Simulator

I began with a base of python code created by Rafael Brune in 2013 which utilizes a class-based architecture and the Poisson nature of block creation to mimic the nodes of participants in the bitcoin network, connections between these nodes (the P2P network), network latency, and the selfish mining strategy [14]. Due credit goes to Brune for creating a well-written simulator that acted a solid base upon which to build [14]. However, there were some errors in the code and limitations of its functionality that required significant alterations to cater to the specifications of my tests, as would be expected in utilitizng any code base. We will now discuss details of the simulator and all necessary alterations.

4.2.1 Overview of the Simulator

The simulator code is composed primarily of classes and functions, which mimic the different types of participants and state changes that can occur in the true Bitcoin network. To name a few examples, there is class for an honest miner and one for a selfish miner, and there are functions for mining and broadcasting blocks across the network. The algorithm used in the selfish miner class is the same as the one referenced in my model section, but implemented in Python. In other words, a selfish miner acting in this simulator follows the same decision process as described in the section on the selfish mining strategy—this is obviously critical, because in order to compare the simulation results to the expected revenue results in the previous section, there must be an exact match between the strategy in the code and the one described in this paper.

The code base contains a script that, combining the functionality of many classes and functions, initializes a set of miners in the network, simulates their efforts to mine blocks, and subsequently keeps track of the blockchain as it grows over time. The simulator sets up random links between different miners in the network such that broadcasting of blocks travels through the network in the same way it would in the actual bitcoin network—namely, the simulator mimics the P2P network. In creating these random links between nodes, latency values are assigned to each link, which indicate the number of milliseconds it takes for a block to get sent from the miner at one end of the link to the miner at the other end. It should be noted that this simulator does not mimic the process that true miners go through in the proof-of-work process to mine a block. Rather, the code uses a random generator to simulate block creation via a Poisson process, which mirrors the frequency and spacing between block discovery in the true network.[38] This mining process difference between the true Bitcoin network and the simulator is certainly worth noting but does not seem to have any impact on the statistics that I am outputting from the simulations.

This code base contained errors when I first obtained it, and it was not fully equipped to handle the simulations necessary for investigating the questions this paper presents. Thus, I altered the existing code to fix bugs, deleted code that was either unnecessary or

[38]Refer to Section 2.3.3 for a reminder on the details of this Poisson process.

problematic, and added code to fulfill the specifications required to answer the questions posed in this paper. Most importantly, the code was configured to only simulate one selfish miner—I restructured it to be able to handle different numbers of individually-acting selfish miners, each with a specified hashing power. Additionally, I added logic to the code that allowed me to extract additional information from the simulations like stale block rates and group-specific revenues. Finally, I built a script that output a spreadsheet of simulation statistics for later data analysis.

4.3 Defining Relevant Variables

Before discussing the simulations that were run, we need build a clear mapping between the variables relevant to our simulations and the ones used in the Model section. Specifically, we deal with α and ϕ in defining the profitability of the selfish mining strategy in the Model section of the paper, so it is important to discuss how those variables will be incorporated into (or controlled by) the simulator.

4.3.1 α and β

In modeling the selfish mining strategy, we defined α as the fraction of the network colluding to engage in that strategy. This also meant that the rest of the network, composing $1-\alpha$ fraction of hashing power in the network, were assumed to be honest miners. Now that we are de-generalizing the model, such that there will be N selfish mining groups, rather than just one, we need to define new variables. Namely, we will still use α to denote the hashing power of each selfish mining group. However, all N of these selfish miners will in total compose a total β fraction of the network hashing power, while the honest miners will compose the remaining $1-\beta$ fraction. Formalizing that notion, we have

$$\beta = \sum_{i=1}^{N} \alpha_i = N\alpha \tag{16}$$

Notice that in constructing the variable β, we assume that all of the N selfish miners posses an identical α fraction of network hashing power. This is intentional: by partitioning the selfish miners into uniform-sized mining groups, we simplify later analysis. Additionally, it allows us to isolate the effects of a certain number, N, of miners without the confounding factor of varying-sized groups.

4.3.2 ϕ and ω

In modeling the selfish mining strategy, we defined ϕ to be the fraction of miners in the network that mined on the block released by the selfish miner in the case of a bifurcation race between two one-block chains at the same height in the main chain. In this simulator, especially because there will be instances of multifurcation races (more than two blocks of the same height, only one of which will prevail in the race) and complicated dynamics due to the many selfiush miners in the network, it will be unfeasible to artificially set the parameter ϕ within our simulator.[39] What can be done instead is use network latency as a proxy for ϕ.

Network latency is often defined as the number of milliseconds it takes for blocks to move between nodes—high latency means that block propagation from one to another is slow, whereas low latency means this propagation is fast. In the case of a bifurcation (or trifurcation, quadfircation,...) race, miners will mine on the block that arrives at their node first as we discussed earlier. Let's say a particular selfish miner has a lot of low-latency connections, meaning that his block will travel quickly throughout the P2P network. Let's also say a particular honest miner that broadcast the block which initiated the bifurcation race with the selfish miner has a lot of high latency connections. In a bifurcation race involving the two aforementioned theoretical miners (one honest and one selfish), the selfish miner's block would reach many nodes before that of the honest miner—and thus, more miners would mine on top of the selfish miner's block rather than that of the honest miner. This example illustrates the fact that the fraction of miners who mine upon the selfish miner's block, defined as ϕ in the model section, is closely related to network latency. Thus, we can use network latency of selfish miners (compared to that of honest miners) as a proxy

[39]As we will soon see, when there are many selfish miners in the network, "multifurcations" can occur, which are races between more than two broadcasted blocks at the same height. And because ϕ is a variable designed specifically for only two blocks at the same height, bifurcations, using ϕ in our model would not even work nor make sense.

for ϕ.

To formalize our proxy for ϕ, we will let ω represent the latency between two nodes. The support of this variable is $(0, \infty)$, measured in seconds: near-zero latency means sending a block from one node to another is nearly instantaneous, whereas as higher values means it takes a long time for this send-and-receive to occur. Because each node in the true bitcoin network has hundreds of connections (at varying latencies), and each of those nodes similarly has many connections, building an exact relationship between ϕ and ω is nearly impossible. We can, however, build a general relationship between the two variables. Specifically, this relationship is an inverse one: lower values of latency between a selfish miner's node and its connections implies a higher value of ϕ, and higher values of latency between a selfish miner's node and its connections implies a lower value of ϕ. In our simulations, when we are looking at the effects of latency, as a proxy for ϕ, on selfish miner revenues, we will vary the ω for the connections of the selfish miners while keeping the (average) ω for the connections of honest miners constant. As we will soon see, the differences between ϕ and ω turn out to contribute to large differentials between the model results and the simulated results.

4.4 Conducting Simulations

The primary goal of our simulations is to analyze the effects of the number of selfish miners in the Bitcoin network on the relative revenues of those miners (as well as the revenues of the honest miners). To acheive this goal, we will run simulations in which we vary the β values and N values of the sum shown in Equation 16. This will in turn allow us to vizualize how N selfish mining groups of hashing power fraction α will impact the revenues of eachother.

By varying both β and N, we will be able to see the dynamics that occur when there are a lot of particularly substantial selfish mining groups in the network (i.e. 3 mining groups, each with over 20% of the hashing power of the network) and those that occur when there are a lot of smaller selfish mining groups (i.e. 25 groups, each with 0.8% of the network hashing power). Specifically, we will run simulations for $\beta \in \{0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1\}$ and $N \in \{1, 2, 3, 4, 5, 10, 25, 50\}$, and we will visualize and analyze all combinations of these β and N. Each simulation will simulate two-and-a-half weeks of the Bitcoin network, meaning that on average 2520 blocks will be mined in each simulation run.[40] Furthermore, we will run three such simulations: each will vary ω, as a proxy for ϕ, to see how this factor affects the results. Specifically, we will run one for $\omega = 0$, another for $\omega = 15$, and the other for a ω value consistent with those assigned to honest miners. In the fist case, we are trying to proxy $\phi \approx 1$, in the second case we are trying to proxy $\phi \approx 0$, and in the final case we are trying to proxy $\phi \approx 0.5$. Note that in the code, we assign ω to each honest miner as a uniformly-chosen random value on the interval $[0.02, 0.22]$, which is an arbitrary choice but one that mirrors feasible network latencies. Keep in mind that it is not very relevant the particular ω value we assign to the connections of honest miners—it is just important that the ω we choose for selfish miners compared to those we assign to honest miners proxies the the value of ϕ we want for a given simulation run.

Now that we have discussed the parameters of the simulations we will be running, we can discuss the outputs of the simulations. These outputs will require transformations to allow us to answer the questions posed in this paper.

4.5 Transforming Simulation Output

Most of the raw output from our simulations on its own is not particularly meaningful to the questions at hand—we need to transform it for a one-to-one comparison with the model results. Specifically, we want to answer the following question: What is the difference in relative revenue of a selfish miner with an α fraction of the network hashing power when he/she is the only selfish miner in the network versus his/her revenue if there are other selfish miners in the network. Essentially, we are testing the robustness of the selfish mining model—with such simple assumptions (i.e. there exits only one colluding selfish mining group in the network) how well can this model predict relative revenues of participants when the network contains many adversaries?[41] In order to perform this type of analysis on the model using the simulation results, we need to transform the simulation output as follows.

[40]2.5 weeks = 25200 minutes $\approx$ 2520 blocks on average over a period of that length, because of the average 1 block mined per 10 minutes property.

[41]Whenever we refer to "adversaries" in this paper, we are simply referring to selfish miners. They are adversaries in the sense that their strategy subverts the rules/protocol of the network.

For any given β and N in the results of the simulations, let's start by understanding what the output on its own will represent. Let's take $\beta = 0.6$ and $N = 5$ as an example. The simulator output for these two values will be some fraction representing the the combined relative revenues of all $N = 5$ selfish miners.[42] Each of these $N = 5$ selfish miners wields a $\frac{\beta}{N} = \frac{0.6}{5} = \alpha = 0.12$ fraction of network hashing power. We are most interested in the relative revenue of each selfish miner (on average), so we will divide the output relative revenue by N. Thus, we will arrive at the average relative revenue of each selfish miner with α fraction of the network hashing power. That is, we are interested in $\overline{r}_{self,rel,sim,\alpha}$. We utilize the same subscript scheme as with our previous definitions involving relative revenue, like in Equation 10 and 14—we note that *sim* denotes a simulation result rather than model result.

Now that we have defined $\overline{r}_{self,rel,sim,\alpha}$, we can make a direct comparison to $E(r_{self,rel,model,\alpha})$ from Equation 14. These two variables are the ones we want to compare for the following reason: we want to see how a selfish miner with hashing power α performs in a network with $N-1$ other selfish miners (each also with α hashing power, for simplicity) compared to how the theoretical model would predict him/her to perform with none of the other $N-1$ selfish miners in the network. We will define one final variable to compare the two quantities, $\overline{r}_{self,rel,sim,\alpha}$ and $E(r_{self,rel,model,\alpha})$: $\theta_{\alpha,N}$ will be the percentage increase of relative revenue with other selfish miners in the network for a given α and N, and thus a $\beta = N\alpha$, over the expectation of relative revenue for a miner with α hashing power in a network with no other selfish miners. Thus, $\theta_{\alpha,N} = 75$ would mean that a selfish miner with $\frac{\beta}{N} = \alpha$ hashing power would have a relative revenue 75% greater in a network with $N-1$ other selfish miners than it would in a network where it's the only selfish miner. We formally define $\theta_{\alpha,N}$ as follows:

$$\theta_{\alpha,N} = \frac{\overline{r}_{self,rel,sim,\alpha} - E(r_{self,rel,model,\alpha})}{E(r_{self,rel,model,\alpha})} \tag{17}$$

where $E(r_{self,rel,model,\alpha})$ is calculated for a given α and ϕ in Equation 14.

[42]As a reminder, relative revenue (over an interval of time) for a miner or mining group is the number of its mined blocks that end up on the main chain as a fraction of the total number of blocks that were mined on the main chain in that time interval.

Graphing $\theta_{\alpha,N}$ as a function of α and N, as well as performing regressions among these variables, will allow us to tease out a relationship between relative revenues of selfish miners in more realistic network dynamics/conditions compared to the results of the theoretical, one-selfish-miner model. We can also see how the relationships of these variables change when network latency, ω, is altered—this will prove to be an important part of our analysis.

Finally, we will perform a similar comparison for the effect on the relative revenue of honest miners by comparing the case in which there are many selfish miners in the network with the simple case of just one selfish miner in the network. We will do this by taking the aggregate relative revenue for honest miners in the simulation and compare it to their aggregate relative revenue in a network where all β hashing power is consumed by one large selfish mining pool (rather than many individual selfish miners aggregating to a total of β). It's critical to note that this comparison is not exactly analogous to the one we are doing for selfish miners. To be clear, in the selfish miner comparison case, we are comparing selfish revenue of one selfish miner as predicted by the model compared to his/her revenue with many other selfish miners in the network . In the case of the honest miners, we are comparing how they are affected when there is one large miner with fraction β of the network hashing power compared to the case where there are N selfish miners, each with its own α fraction of hashing power. So, in the case of analyzing the honest mining revenue, we want to see how they perform when there is a larger colluding selfish mining group compared to how they perform when there are many smaller groups colluding separately. We can similarly look at the honest miner results for different network latency values, ω.

Now that we have built a construct for interpreting simulation output and how to transform it, we will discuss the results.

4.6 Results

We will analyze the relationship between the simulator results and the results that would be expected based on the basic selfish mining model. In this section, we will divide our explanation into three sections, one for each different value of network latency, ω. For each of the latency values, we will focus on three types of graphs: $\theta_{\alpha,N}$ versus N, with differently-colored lines representing each β value for all N, one for $N \leq 5$ (to focus in on the case of a few larger selfish mining groups), and finally $\theta_{\alpha,N}$ versus α (purposely leaving N out of the analysis for this last one).

4.6.1 Results for the Base ω Value

The following graph is the result of the simulation at the value of ω that proxies $\phi \approx 0.5$. In order to proxy $\phi \approx 0.5$, we want to create a situation in which both selfish and honest miners average the same ω, such that ϕ will fluctuate somewhere around 0.5. Thus, since we assign honest miners $\omega = 0.02 + 0.2X,\ \textit{where } X \sim \text{Unif}(0,1)$, we assign selfish miners the same ω.[43] We consider this the base model and most realistic in terms of representing the reality of the true bitcoin network. In the true bitcoin network, with so many participants of varying latency links, we can expect that ϕ would generally be equal to 0.5, and thus our ω generation for each selfish miner is an apt way to proxy that value of ϕ.

The results of this simulation are shown in Figure 15, and we notice a few particularly interesting trends worth discussing. At β values of 0.9 and 0.8, having more selfish miners in the network can increase the relative revenue of any one of those N miners by up to 50% over what the selfish mining model would suggest. On the other hand, for values of β less than 0.7, the selfish miners in the network generally hurt each other as N grows. This implies that if there are a lot of individually-participating selfish miners in the network and selfish miners as a whole compose 70% or less of the total network hashing power, selfish miners will perform worse given their α than the selfish mining model would suggest. Selfish mining is already a strategy that can be very risky when the miner doesn't have a large α, but this result is even more discouraging for prospective selfish miners: the participation of selfish miners hurts the performance of eachother when they compose less than 70% of the network. And in the true bitcoin network, there is no evidence to suggest that there currently exists rampant selfish mining such that that over 70% of the network is engaged in these strategies. Thus, for all intensive purposes, at mid-range ω and thus mid-range ϕ

[43]X is chosen randomly for each selfish (and honest) miner. Also, Unif denotes the Uniform distribution.

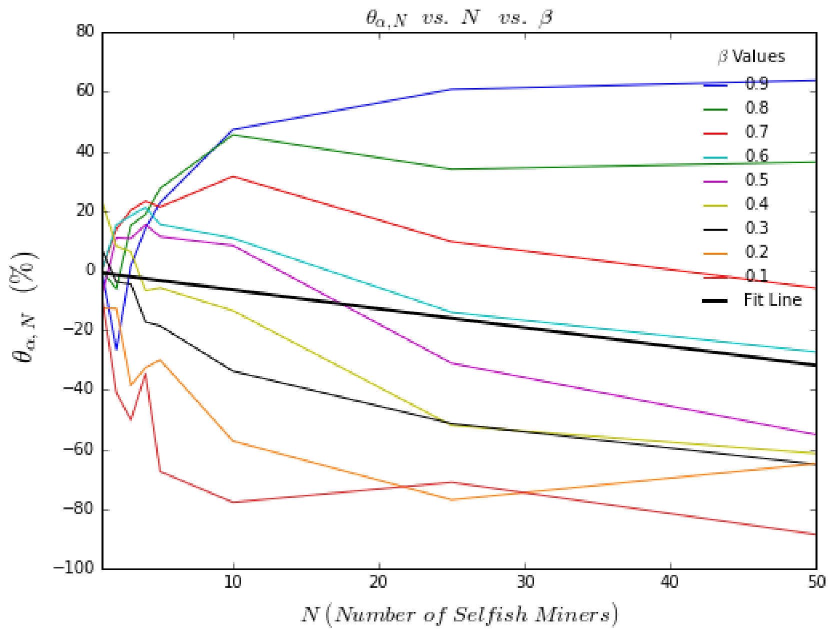


Figure 15: For a base latency value, we that the simulation revenues generally are below those that the theoretical model would predict.

values, selfish miners should understand that their relative revenue can be between a few percentage points lower than the selfish mining model would suggest, and up to 80% lower if the miners have particularly low α values.

Finally, we regress $\theta_{\alpha,N}$ on N (disregarding β) to plot an Ordinary Least Squares line on the graph. This line has a very low R-squared value (0.09), as would be expected due to the highly-varying lines of data that compose the graph. However, this low R-squared value is not a problem, because we are only using the line to provide a visual trend of $\theta_{\alpha,N}$ vs. N across β. Thus, we do not aim to use this line for formal prediction, but rather as a visual summary of the data. [44]

Now, let's take a look at a specific portion of the graph in Figure 16, where $N \leq 5$. This is an important portion of the graph to look at for the following reason: one of the most

[44]In the subsequent sections for the other ω values, we will provide an Ordinary Least Squares regression line as we did in Figure 15. The R-squared values for the OLS lines in the next two sections are 0.14 and 0.27, respectively. We will not explicitly discuss the regression lines again for the rest of the paper—again, they serve as a visual trend rather than a prediction device.

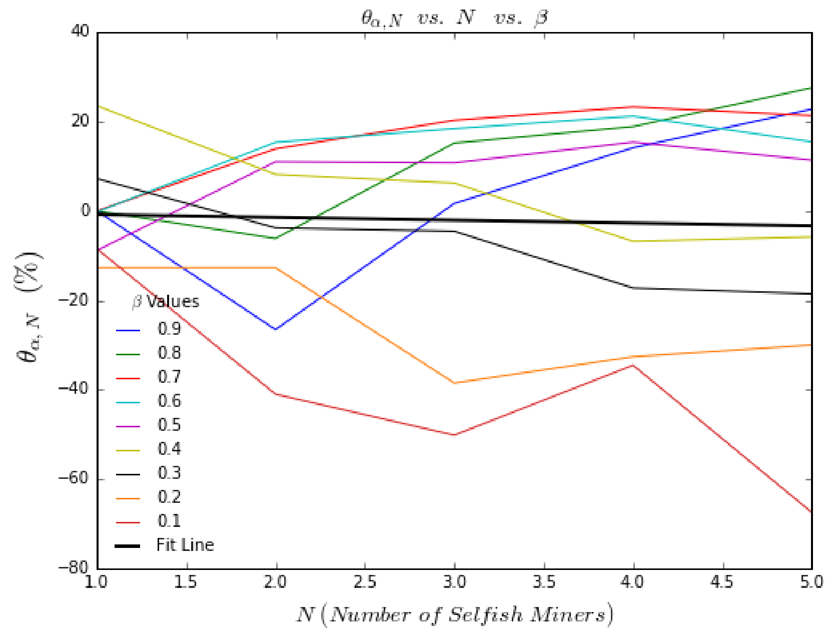


Figure 16: For a base latency value and $N \leq 5$ selfish miners, relative revenues of selfish miners tend to be higher than those that the model would predict.

critical results from the general selfish mining strategy is that, given a medium ϕ value, one generally requires at least $\alpha \geq 0.25$ for the strategy to be profitable over the honest mining strategy.[45] Thus, by looking at the results for $N \leq 5$, we can understand the dynamics of fewer yet larger selfish miners. The graph in Figure 16 indicates that at low values of N, miners with $\alpha \geq\sim 0.1$ can make between 10% and 30% more relative revenue when only a few other selfish miners are participating in the network. This is telling result: in the current bitcoin network, in which there are a few mining pools with $\alpha \geq 0.1$, this information is useful. If these mining pools sense that other pools of similar size are engaging in the selfish mining strategy, they can each expect greater relative revenues than the selfish mining model would suggest for their α. This along with the other results of this section so far are starting to tell a story: the selfish mining model does not seem to be robust enough on its own to account for the complex effects of many selfish miners competing in the system.

[45]See section on selfish mining revenues

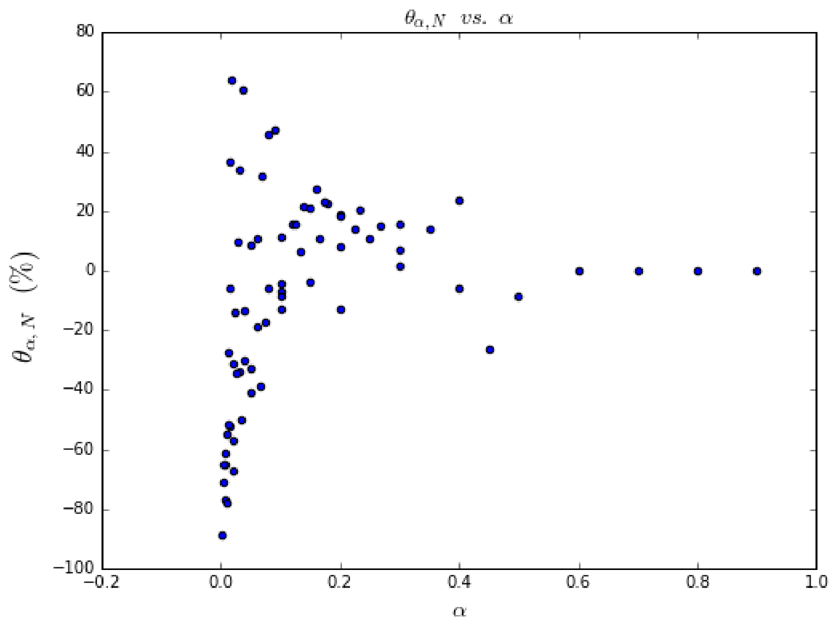


Figure 17: For a base latency value, we see that the relative revenues from the simulations in comparison to relative revenues that the model would predict are spread for low values of α but converge to ~ 0 as α increases

Let's now analyze the final graph of this subsection, which can be seen in Figure 17. It is important to note that this graph is very different from the previous graphs in a few ways: first of all, the x-axis is no longer N, but rather α. Also, we do not visually differentiate between different N values in this graph. Rather, this graph gives us a general trend: despite the exact number of other selfish miners in the network, we can see how alpha values generally relate to the the model's expected relative revenue when at least some other selfish miners exist in the network. This graph conveys the idea that, at low values of α, there can be wide variations in the increase (or decrease) in relative revenue over the selfish mining model results—but, as α grows, the relative revenues approach the expected relative revenue from the selfish mining model. Note that the dots near $\theta_{\alpha,N} = 0$, corresponding with $\alpha \in \{0.6, 0.7, 0.8, 0.9\}$, represent the results of $\beta \in \{0.6, 0.7, 0.8, 0.9\}$ where $N = 1$. As we described earlier in the paper, whenever a miner has control over 50% of the network, selfish or otherwise, they essentially control the main blockchain and can take almost all of the revenue in the system.[46] Thus, because the model and simulation both account for the 51% attack scenario, we see $\theta_{\alpha,N} = 0$ for all data points in which a miner possesses over 50% of the hashing power in the network.

In the Appendix, we present graphs for $\theta_{\alpha,N}$ vs. α (rather than vs. $\theta_{\alpha,N}$ vs. N vs. β) for the ω value discussed in this section, and for the two ω values discussed in each of the subsequent sections. These graphs represent the exact same data, but in a different form. We will not discuss those graphs explicitly, but they ultimately encode the same information that the $\theta_{\alpha,N}$ vs. N vs. β graphs do, and can help to elucidate understanding of the general trends of each variable.

4.6.2 Results for $\omega = 0$

Let's now look at how the results change when we alter the latency of the node connections of selfish miners—specifically, we look at the case of $\omega = 0$. The first graph used in this section's analysis can be found in Figure 18. Here we see a consistent pattern: low latency among the selfish miners actually hurts the selfish miners as a whole. At first this may seem counter-intuitive: when latency in the network is low, selfish miners are very likely to win bifurcation races ($\phi \approx 1$), thus mining more blocks and having higher overall revenue. However, when we refer back to the relationship between ω and ϕ, the appearance of this graph becomes more understandable. Superficially, even setting $\omega = 0$, the honest miners in the network have varying latency connections as we discussed earlier, meaning that selfish miners will not necessarily win every bifurcation race. In other words, just because $\omega = 0$, it is most likely not true that $\phi = 1$. In fact, it could even correspond to much lower values of ϕ because of the sheer size and randomness of the connections in the network. Thus, selfish mining model for $\phi \approx 1$ will generally overestimate the relative revenues of selfish miners. As shown by our graph, this difference is accentuated for lower values of β and higher values of N, because there will be more multifurcation races with more selfiush miners, causing a lot of lost potential revenue for selfish miners in these situations [47]

[46]See the Section "51% Attacks."

[47]"Multifurcations" are the generalized version of bifurcations. This means that rather than just two blocks at the same height in the main chain fighting to prevail, there will be multiple. We save discussion of this situation for the Discussion section.

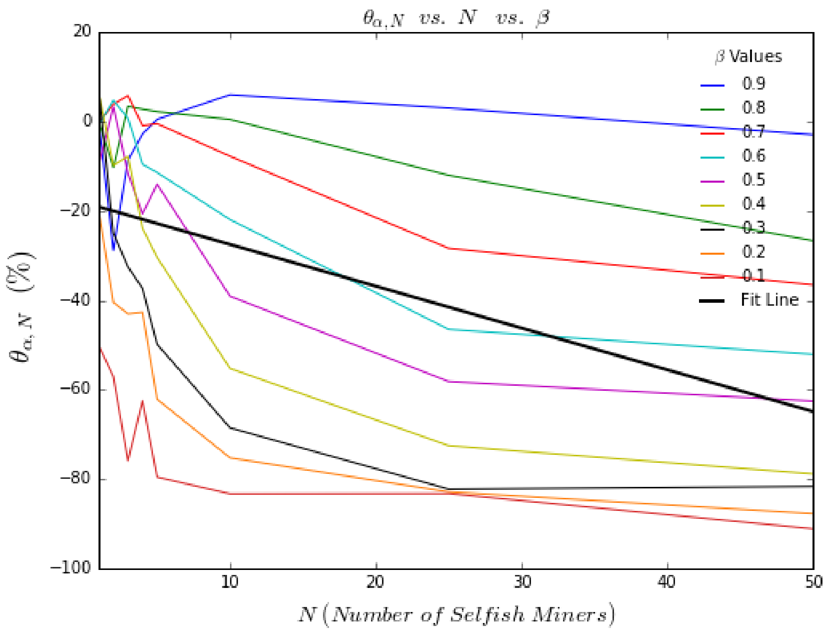


Figure 18: At low latency values, the simulation relative revenue results are generally well below the relative revenues predicted by the selfish mining models.

Zooming in on the case for $N \leq 5$ in Figure 19, we see a similar result as with the zoomed-out graph–however, the simulation results are closer to those that the selfish mining model would predict. This seems to make sense because at lower values of N, the differential between ω and ϕ is less exaggerated. And thus, we can say that for low values of N, $\theta_{\alpha,N}$ is only slightly negative for most β. We finally take a look at the graph of α vs. $\theta_{\alpha,N}$ in Figure 20 for the current ω under investigation, $\omega = 0$. We see that at low values of α (which by definition correspond with high values of N, low values of β, or both), relative revenues from the simulation fall well below those of the model. This corresponds with our results from the earlier graphs in this subsection. However, we see that at higher values of α, the simulator relative revenues converge to the revenues produced by the selfish mining model. This is not unexpected because, even at low latencies, latency becomes less important at higher α values (in both the selfish mining model and the simulations). When a selfish miner has a high α, he/she can contribute significantly to ϕ, because that miner

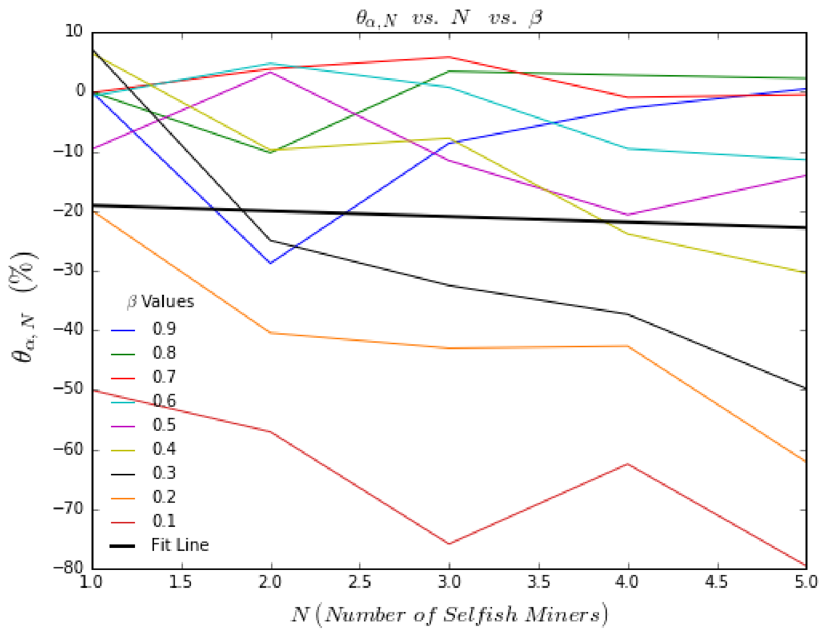


Figure 19: At low latency values, with $N \leq 5$, we still see that the simulation relative revenue results are generally below the relative revenues predicted by the selfish mining model.

can put all of his/her relatively large α towards his/her own block in a bifurcation race. This is accounted for in the model and occurs naturally in the simulator, so we expect to see the convergence that we do in the graph to $\theta_{\alpha,N} = 0$ for higher α.

4.6.3 Results for $\omega = 15$

Finally, we look at the situation for a high value of ω. The results show that, in general, the simulator indicates much higher relative revenues than the selfish mining model would predict. These results align with intuition for the following reason: when selfish miners have high latency connections, ϕ is likely to be very low, meaning that selfish miners will almost always lose bifurcation races. But what the basic model doesn't account for is that, when there are many miners with high latency, there will be a lot of additional stale blocks (created by the many mutifurcations that will occur in a network of many selfish miners), meaning that the main chain ultimately contains fewer blocks over a given time

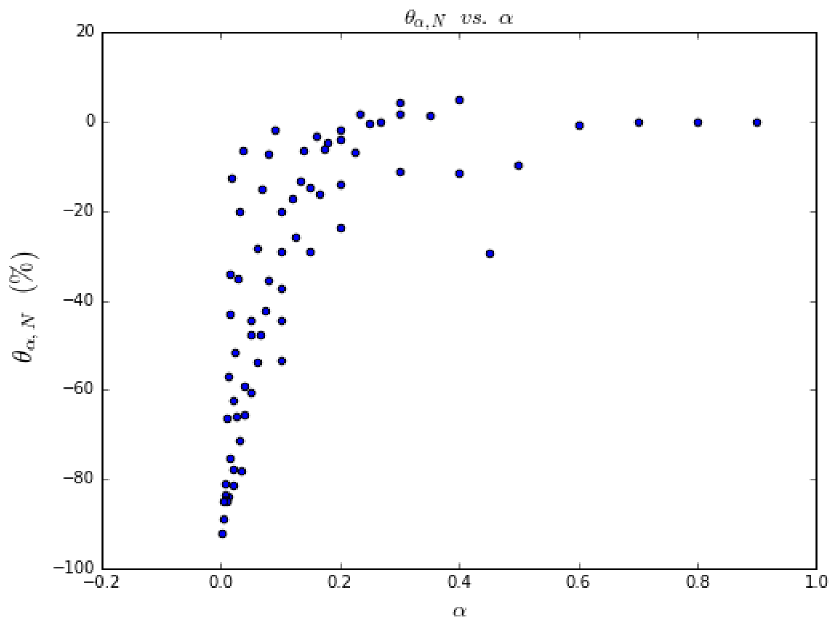


Figure 20: This graph corresponds with the results from our other two graphs for low latency. At low α, simulation relative revenues are well below those predicted by the selfish mining model—however, as α approaches 1, $\theta_{\alpha,N}$ approaches 0.

interval. Because relative revenue is the number of blocks found by a miner divided by the total number of blocks in the main chain, a shorter main chain can mean a much higher relative revenue. Thus, because the selfish mining model does not take into account the many additional stale blocks created by N selfish miners, we expect the relative revenues from the simulation results to be well above those expected from the model. With the exception of some values of lower N and high β, we see the similar results when we zoom in to $N \leq 5$ as we do with the graph for all N values (see Figure 22). In a few cases, like for $\beta \in \{0.9, 0.1\}$ and $N \in \{2, 3\}$, we see $\theta_{\alpha,N}$ drop below 0, but these results may not have an easy explanation. Because these simulations contain many sources of randomness (Poisson process block generation, random latency assignments for honest miners, etc.), there are sure to be some results that are not easily explicable—this is one of such results. Finally, we look at the α vs. $\theta_{\alpha,N}$ graph in Figure 23 for the case of $\omega = 15$. At low values of α, we actually see both negative and positive values of $\theta_{\alpha,N}$, although the majority fall in

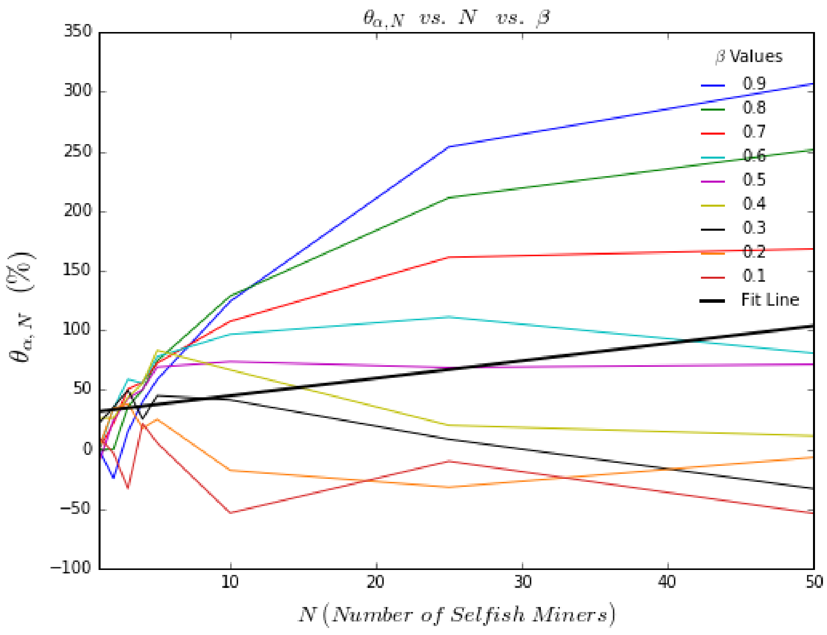


Figure 21: At high latency values, the simulation relative revenues are sometimes greatly underestimated by the selfish mining model and other times slightly overestimated by the selfish mining model. Specifically, at higher values of β, we spot the aforementioned underestimation by the selfish mining model.

the positve territory. As with the corresponding graph for the other two ω values, we see a smilar narrowing of the spread of $\theta_{\alpha,N}$ as α grows, eventually converging to 0. At low α, we see large spread and a skew towards positive $\theta_{\alpha,N}$, corresponding with our results from the previous graphs for $\omega = 15$. The convergence to near-zero values for high α reinforces the point made in the previous subsection: when there are fewer and larger selfish miners, the distortions in relative revenue from high numbers of stale blocks become less significant when these miners can essentially override the effects of ω because of their sheet size (and thus power) in the context of the network.

4.6.4 Effects on the Honest Miners

One of the core tenets of the original selfish mining models is that honest miners lose out at the expense of selfish miners, depending heavily on the α of the selfish miner in the network.

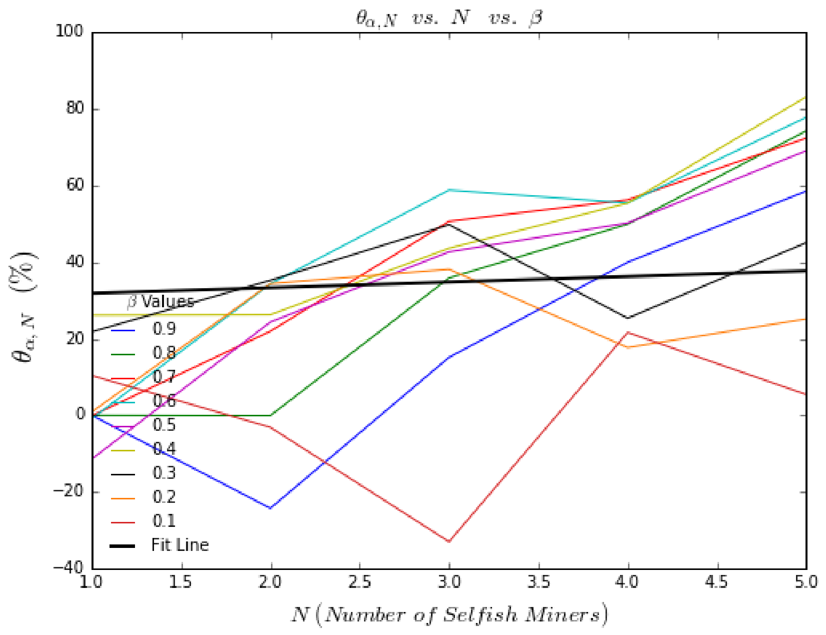


Figure 22: When we zoom in to $N \leq 5$ for high latency, we see that the simulation relative revenues are still almost always greater than what the selfish mining model would suggest.

In the case of many selfish miners rather than just one, an important question is as follows: does having N selfish miners, each with hashing power α, affect the honest miners differently than would having one selfish miner with $\beta = N\alpha$ hashing power in the network? As it turns out, we have already answered this question: in the results section thus far, for each of the three ω, it holds true that the results of $\theta_{\alpha,N}$ for selfish miners are in the opposite direction for the case of honest miners. Because of the way we define relative revenue, this fact is true by definition: honest miners will have $1 - S$ if S is the aggregate relative revenue fraction for all selfish miners in the network. To make this more clear, let's take the case of $\omega = 0$ that we analyzed. We concluded in that section, based on the graphs and discussion, that depending on the β and N values, the model often overestimated the results of the simulator (corresponding to negative values of $\theta_{\alpha,N}$), and sometimes underestimated the results of the simulator (corresponding to positive values of $\theta_{\alpha,N}$). This means that in a network of many selfish miners, selfish miners tend to do worse than the model would predict due to the

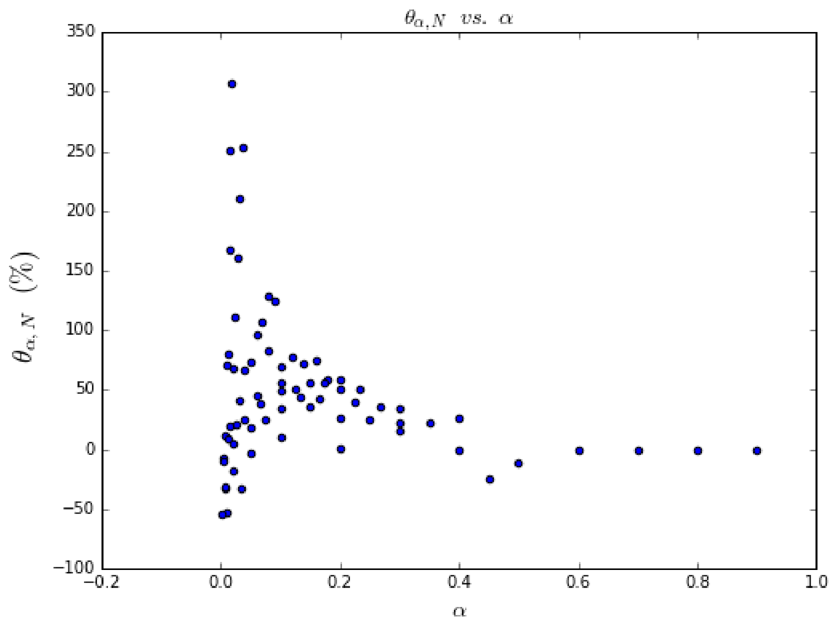


Figure 23: This graph generally corresponds with the results from our other two graphs for high latency. At low α, simulation relative revenues are well above those predicted by the selfish mining model—however, as α approaches 1, $\theta_{\alpha,N}$ approaches 0

effects of the many selfish miners in the network. This in turn means that the opposite is true for honest miners—they would generally perform better than the model would expect. That is, honest miners do better than the model would suggest in some but not all cases, because the many selfish miners essentially dilute each other's relative revenue, leading to higher relative revenues for the aggregate of honest miners. It should be noted, though, that in the case of very high β values, honest miners actually do worse than the model would suggest. The graph shown in Figure 24 presents these results.

Note that the graph is not an exactly inverted version of the corresponding graphs for $\theta_{\alpha,N}$ for the selfish miners. Looking at the formula for $\theta_{\alpha,N}$ and the definition of $\overline{r}_{rel,sim,\alpha,honest}$, and $E(r_{rel,model,\alpha,honest})$, it becomes clear that the graphs for honest miners encode opposite effects of those for selfish miners, but the exact relationship between these graphs is not as simple as S and $1 - S$, for example.

It is not necessary to go through all of the ω cases and graphs for the $\theta_{\alpha,N}$ cor-

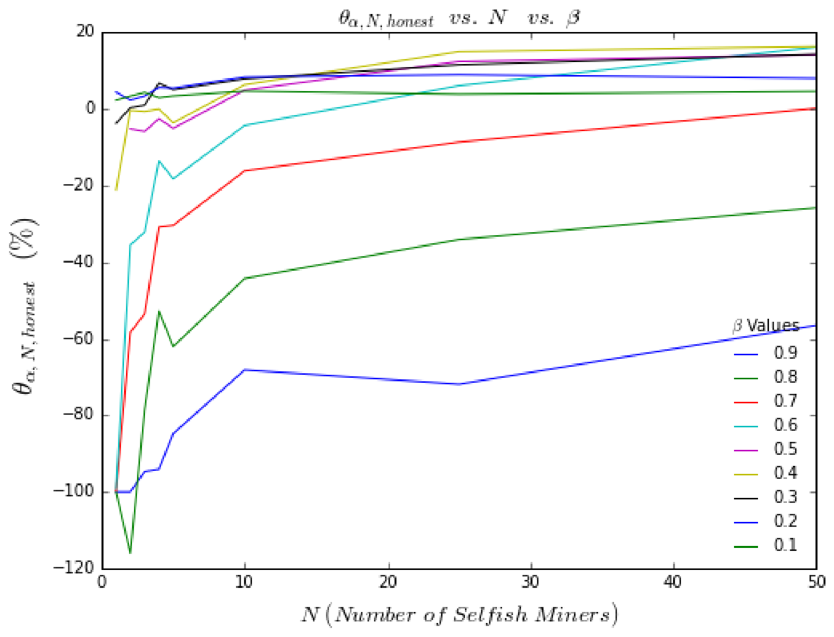


Figure 24: $\theta_{\alpha,N}$ for honest miners at low latency shows that honest miners generally do better than the model would suggest. However, this is not the case for the three highest beta values, in which the effects of such a high density of selfish miners compared to honest miners changes is a strong enough factor to lead to opposite directionality in $\theta_{\alpha,N}$.

responding with honest miner revenue. Rather, the results of the sections for the selfish miners already encode the answers for the model's representation of relative revenues for honest miners versus that of the model: the effect for honest miners is necessarily in the opposite direction the results for the selfish miners. And thus, for low values of latency, the honest miners' relative revenue tends to be underestimated by the model, and for high values of the latency, the honest miner's relative revenue tends to be overestimated by the model. These results are logical, because in the case of low latency, selfish miners accrue less relative revenue than the model would suggest due to the discrepancies between ω and ϕ, meaning that the honest miners have greater relative revenues compared to the model—they actually win bifurcation races more often than the model with $\phi = 1$ would suggest. In the case of the high latency, we know that miners in the simulation are causing so many stale blocks that their relative revenue is higher than the model would suggest, and therefore the honest miners are left with a smaller fraction of the revenue.

4.7 Discussion

4.7.1 Meaning of the Results

The results of our simulations point to a reasonably clear conclusion: the selfish mining revenue model that combines the effects of α and ϕ to determine expected relative revenues of selfish miners is inadequate to account for more realistic cases in which more than one selfish miner participate in the network. This is not to say that the model is not a very useful—as we mentioned earlier, the model does an excellent job predicting relative revenues under the one-selfish-miner scenario. However, our results reveal that the prevalence of selfish miners tends to lead to situations in the blockchain construction process that cannot be accounted for in the selfish miner model in its current state. Possibly the most crucial example of a case that occurs often in a network of many selfish miners that would not happen in the one-selfish-miner case (well, it would be very rare) is what we will call a "multifurcation" race, which we mentioned briefly earlier. We will now discuss the details a this multifurcation race and understand why they are so important.

Let's say there are 4 selfish miners in a network who have each separately mined a block at the same height. Each of them is following the selfish mining strategy: if a block is broadcasted in the network by some other miner, each selfish miner will immediately release his/her private chain of 1 block, initiating a race. However, as soon as this theoretical miner broadcasts his/her block, there will be four (not just one!) selfish miners who nearly-simultaneously broadcast their private chain block to the rest of the network. Now, instead of having a bifurcation race between an honest miner's block and a selfish miners block, we have a pentfurcation race between an honest miner's block and four selfish miners' separate blocks, meaning there will be five blocks, not just two, racing to become part of the main chain. I call the effect of many selfish miners in the network "The cascade effect:" when many selfish miners are present in the network, it is inevitable that many of them will be simultaneously holding private chains, leading to a race of many blocks (hence the term "pent-" or more generally "multi-" furcation race) rather than the typical race of two blocks.

These multifurcations have competing effects on revenues for the selfish miners. On one hand, each selfish miner is less likely to win with more competitors in the race. On the other hand, the combination of many selfish miners makes it less likely for the one honest miners to win the race, which helps the aggregate relative revenue of all selfish miners. Not to mention, the meaning of ϕ from the selfish mining revenue model changes significantly when there are more than one selfish miner involved in a race: each will have its own ϕ, but that ϕ will greatly vary depending on how many blocks are involved in the block race. Additionally, one selfish miner's ϕ takes away from the ϕ of the other selfish miners involved in the race. As we can see from this discussion, multifurcations bring a very interesting yet complex and unpredictable aspect to networks involving many selfish mining groups. While it is hard to definitively claim causality in the the simulations we've run, we can confidently say that the multifurcation races that occur naturally as a result of many selfish miners impacts the results in complex ways.

4.7.2 Comments on α and ϕ

We see some general trends from the results of the simulations that are worth summarizing. The higher fraction of the network hashing power a selfish miner holds, α, the less beholden they are to other variables that affect the network. For example, we saw in every case that higher α values were associated with simulated relative revenues closely mirroring relative revenues as predicted by the selfish mining model. This means that the simplicity of the model becomes more sufficient for selfish miners possessing higher α values. This makes sense because hashing power is ultimately the most important contributor to expected relative revenue in the network: in the long run, higher α leads to more blocks discovered, which subsequently implied higher (relative) revenue.

One of the results of the the selfish mining model, which can be easily arrived at because of its simplicity, is that, for a given ϕ, there exists a threshold of α above which the selfish mining strategy is more profitable than honest mining. Due of the complexity of our simulations, and the nature of the results, defining such cutoffs is much more difficult.

In other words, it would be very helpful if we could use the results of our simulation to update the thresholds of α for a given ϕ when a network has N selfish miners. However, it is important to keep in mind that our simulations were not performed with the goal of finding these cutoffs: rather, our simulations were performed in an effort to see how the theoretical model results hold up under more complex and realistic conditions.

As we saw, ϕ and its proxy, ω, played a signifcant role in our results. In general, as the selfish mining revenue model emphasized and my simulation supports, the importance of ϕ to profitability of the strategy cannot be denied. One of the defining situations of the selfish mining strategy is the bifurcation race (or, as we now know, the multifurcation race!)—subsequently, whether a selfish miner wins this race is critical to his/her relative revenue. The problem, though, is that ϕ is a great variable in theory and helps build a clean model. But, understanding ϕ values in a network is extremely difficult. As we have now discussed in depth, while ω seems to be a reasonable proxy for ϕ, it often is mismatched with ϕ. In fact, one of the most notable takeaways from our simulations is this very reality: ϕ is ultimately a misleading variable for a model because it can be hard to understand in the context of a complex network of miners. Hopefully in future models that assess the profitability of subversive strategies in bitcoin mining, ϕ will be either strongly qualified or replaced with other more-easily-understood (and more realistic) network variables, like ω.

4.7.3 Natural Next Steps

While it is beyond the scope of this paper, a natural next step would be to build an advanced and robust model for selfish miner revenues that can appropriately account for both the behaviours of other miners in the network and influential attributes like network latency. Some papers in this space have expanded the model to account for network parameters like latency, block size, bandwidth of nodes, etc. However, none explicitly account for the game-theoretical nature of a network of strategic participants. In fact, a Markov Decision Process could be used to build a model of such complexity, taking game theory and other network factors into account. In short, Markov Decision Processes "provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker." This type of model would allow us to account for many more of the complexities the bitcoin network than does the current model for selfish mining [15].

4.7.4 Qualifications of Results

As with any study, it is always important to point out potential flaws in the process or results. Furthermore, it is important to point out the variables or circumstances a model or simulation does not account for. In this subsection, we will discuss such potential flaws and simplifications made in this paper.

Bandwidth, which represents the amount of information a link between nodes can "fit", and block size, the literal size in memory of a mined block, are two variables that would affect (as shown by other papers in this space) the propagation of blocks across the network, revenues of miners, and other simulation statistics. Based on the questions posed in this paper, and because I wanted to make a direct comparison between the selfish mining revenue model and the simulations, I left out the manipulations of these two additional network "variables." Thus, instead of manipulating block size and/or bandwidth, our simulations held those variables constant, thus preventing any unexplained or unwanted effects on our results

Another aspect of the study that could be improved upon is the simulator: the true bitcoin network is infinitely complex, and any simulator always has room for improvement in proxying the true network. For our study, I am confident that our simulator mimicked the processes that are most relevant to my questions. Namely, the rate of mining (estimated via Poisson process), the selfish mining algorithm, and the P2P network are all the key aspects required to go about answering the questions posed in this paper. And in these areas, our simulator does a good job at proxying the true bitcoin network.

5 Conclusion

Bitcoin and other cryptocurrencies are constantly evolving—changes to the protocol of each currency are common, and higher rates of use often cause developers to re-think aspects of network design or rules. When bitcoin was created, the mining process was thought to be incentive compatible—strategies other than following the protocol were not thought to be profitable. If a miner has certain threshold of hashing power, and other network conditions in his/her favor, it turned out that subversive strategies, specifically selfish mining, could be profitable. As economic theory would suggest, rational actors in a market aim to maximize their profitability, so it makes sense that mining pools began engaging in this strategy.[] Thus, I sought to see how the well-established selfish mining revenue model would hold up under complex and realistic network conditions.

And while building mathematical models, like the selfish mining revenue model, can be particularly useful in understanding the way that these strategies unfold and return profit, these mathematical models can only be so accurate for an often much-more-complex true networks. This paper not only puts the theoretical model of selfish mining revenue to the test, but it highlights a more important fact about cryptocurrency mining in general: because of network complexities, it is important to simulate as many of the true dynamics of a network as possible to understand the nuanced results that can alter the results of a general mathematical model.

As cryptocurrencies become an integral medium of exchange in many countries, it will be more and more important to understand how potentially adversarial participants in these networks can affect other participants. Along with other papers that simulate various cryptocurrency networks, I hope that this paper helps set a precedent for simulating network conditions to better understand how these networks evolve over time under certain actions of its participants.

References

[1] Andreas M. Antonopoulos, *Mastering Bitcoin: Programming on the Open Blockchain.* O'Reilly Media Inc. 1-149,171-275 (2017)

[2] Bitcoin Wiki, *Private Keys*
`https://en.bitcoin.it/wiki/Private_key`

[3] Ittay Eyal, Emin Gün Sirer. *Majority is not Enough: Bitcoin Mining is Vulnerable.* Department of Computer Science, Cornell University (2013)

[4] Arthur Gervais, Ghassan O. Karame, Karl Wust, Vasileios Glykantzis, Hubert Ritzdorf, Srdjan Capkun. *On the Security and Performance of Proof of Work Blockchains.* Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. Vienna, Austria (2016)

[5] Nicolas T. Courtois, Lear Bahack. *On Subversive Miner Strategies and Block Withholding Attack in Bitcoin Digital Currency.* 1-9. (2014)

[6] Ayelet Sapirshtein, Yonatan Sompolinsky, Aviv Zohar. *Optimal Selfish Mining Strategies in Bitcoin.* School of Engineering and Computer Science, The Hebrew University of Jerusalem, Israel Microsoft Research, Herzliya, Israel. 1-18, 23-29 (2015)

[7] Vivek Kapoor, VS Abraham, R Singh. *Elliptic Curve Cryptography— An Implementation Tutorial* Ubiquity. 1-7. (2008)

[8] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash Cystem.* (2008)

[9] Bitcoin Wiki *Addresses*
`https://en.bitcoin.it/wiki/Address`

[10] R.A. London *Bitcoin's deflation problem.* https://www.economist.com/blogs/freeexchange/2014/04/money. (2014)

[11] John Carl Villanueva. *How Many Atoms Are There in the Universe?.* https://www.universetoday.com/. (2009)

[12] Joe K. Blitzstein, Jessica Hwang *Introduction to Probability* Chapman and Hall/CRC. 217-219, 222-227,C (2014)

[13] Joshua A. Kroll, Ian C. Davey, and Edward W. Felten *The Economics of Bitcoin Mining,or Bitcoin in the Presence of Adversaries*. Princeton University. (2013)

[14] Rafael Brune *Bitcoin Network Simulator*. https://github.com/rbrune/btcsim. (2013)

[15] Markov Decision Process *https://en.wikipedia.org/wiki* $/Markov_decision_process.Wikipedia, thefreeencyclopedia$(2018)

[16] Blockchain.com *https://www.blockchain.com/*.BLOCKCHAIN LUXEMBOURG S.A.

A1

We show the solution first for s_1, and then for each other the other s_i. We notice that the infinite series is a geometric series, so we can reduce it to $\dfrac{1}{1-(\frac{\alpha}{1-\alpha})}$ in the subsequent calculations.

$$\begin{aligned}
s_1 - \alpha s_1 + \frac{1}{\alpha} s_1 + \sum_{i=2}^{\infty} \left[\left(\frac{\alpha}{1-\alpha} \right)^{i-1} \right] s_1 &= 1 \\
= s_1 (1 - \alpha + \frac{1}{\alpha} + \frac{1}{1-(\frac{\alpha}{1-\alpha})}) &= 1 \\
\text{So, } s_1 &= \frac{1}{1-\alpha+\frac{1}{\alpha}+\dfrac{1}{1-(\frac{\alpha}{1-\alpha})}} \\
s_1 &= \frac{\alpha - 2\alpha^2}{2\alpha^3 - 4\alpha^2 + 1}
\end{aligned}$$

Using the fact that $s_0 = \frac{1}{\alpha} s_1$, we can say that:

$$s_0 = \frac{\alpha - 2\alpha^2}{\alpha(2\alpha^3 - 4\alpha^2 + 1)}$$

Using the fact that $s_{0'} = (1-\alpha)s_1$, we can say that:

$$s_{0'} = \frac{(1-\alpha)(\alpha - 2\alpha^2)}{2\alpha^3 - 4\alpha^2 + 1}$$

And finally, we can easily generalize the steady state probability $\forall i \geq 2$ by relating s_i to s_1:

$$\forall i \geq 2 : s_i = \left(\frac{\alpha}{1-\alpha} \right)^{i-1} \frac{\alpha - 2\alpha^2}{2\alpha^3 - 4\alpha^2 + 1}$$

A2

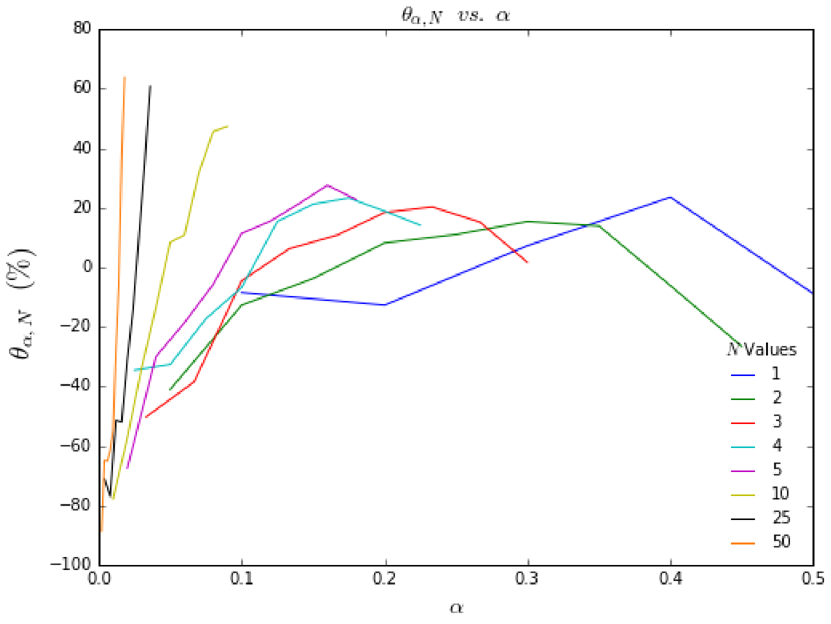


Figure 25: This graph serves as another way to vizualize the trends for the base latency case discussed in section 4.6.1.

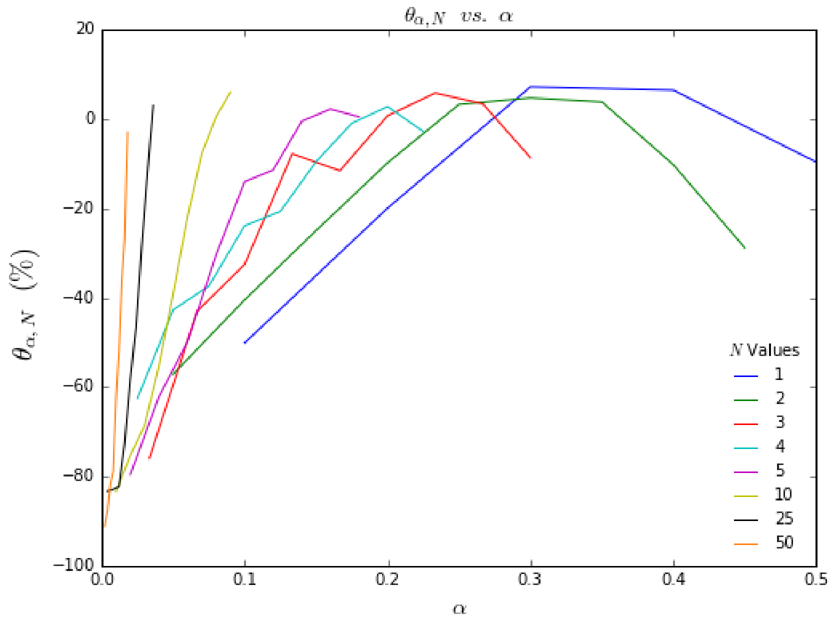


Figure 26: This graph serves as another way to vizualize the trends for the low latency case discussed in section 4.6.2

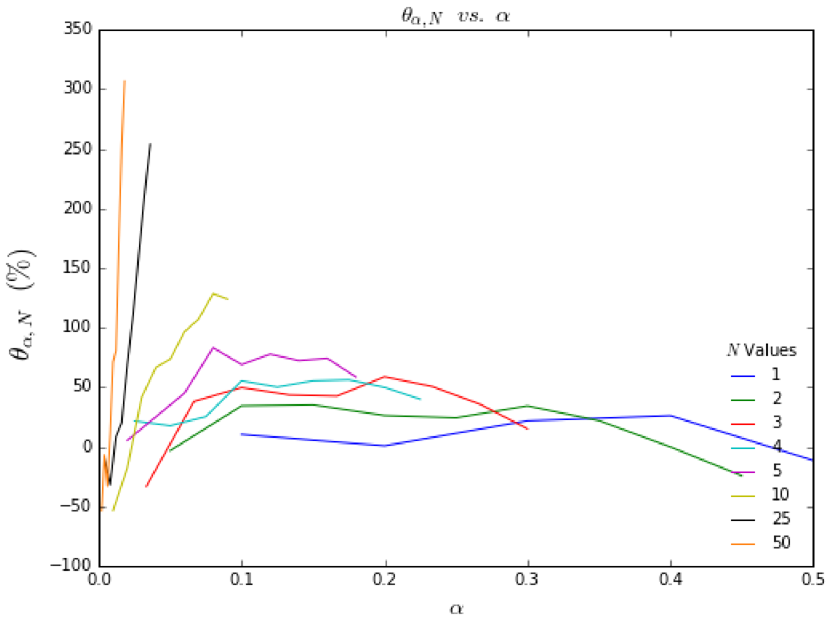


Figure 27: This graph serves as another way to vizualize the trends for the high latency case discussed in section 4.6.3.