QUANTIFYING AND IMPROVING THE PERFORMANCE OF BLOCKCHAIN SYSTEMS

A Dissertation Outline Presented

by

A. PINAR OZISIK

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2019

College of Information and Computer Sciences

QUANTIFYING AND IMPROVING THE PERFORMANCE OF BLOCKCHAIN SYSTEMS

A Dissertation Outline Presented

by

A. PINAR OZISIK

Approved as to style and content by:

Brian N. Levine, Chair

Philip S. Thomas, Member

Yuriy Brun, Member

Nikunj Kapadia, Member

James Allan, Chair
College of Information and Computer Sciences

ABSTRACT

QUANTIFYING AND IMPROVING THE PERFORMANCE OF BLOCKCHAIN SYSTEMS

MAY 2019

A. PINAR OZISIK

B.Sc., BRANDEIS UNIVERSITY

M.S., UNIVERSITY OF MASSACHUSETTS

Directed by: Professor Brian N. Levine

In this dissertation, we analyze and improve the security and performance of blockchain systems across three primary themes. In the first theme, we analyze Bitcoin Core's algorithm for setting difficulty, a network parameter that controls the inter-arrival of blocks. Fluctuations in mining power can cause uneven inter-block delays when the difficulty is not set accurately. Mining power can change due to many reasons, including the miners' allocation of hardware and swings in the exchange rate of a currency. For example, Bitcoin Cash saw enormous variance in mining power at its creation and the algorithm for difficulty did not easily converge. Therefore, we propose and characterize two alternatives to accurately update difficulty: one that solely uses information that is currently available, and another based on status reports that are partial blocks regularly broadcast.

Status reports add overhead into networks because they require the propagation of additional information. In a second theme, we introduce a novel method for the prop-

agation of status reports and blocks. We show that our approach, called Graphene, improves network performance by reducing the size of blocks.

In the third theme, we analyze the practical feasibility of prominent attacks, such as double spending and selfish mining, on blockchain systems. Most analyses generally assume that the mining power of honest and malicious miners is known by an attacker. However, we show that estimation of mining power introduces error into these models. Therefore, we argue that these attacks are difficult to carry out with high precision, and use reinforcement learning techniques to realistically evaluate them when an attacker does not have full knowledge of the network's mining power.

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

A growing number of people have turned to decentralized *virtual currencies* (VCs) such as Bitcoin Core [21], Litecoin (`http://litecoin.org/`), Zerocash [28] and Ethereum [11] for convenience, speculation, or as a potential source of financial privacy and security. The benefits of virtual currencies are many: low transaction fees, transactions over the Internet, and potentially, convenience and privacy.

Centralized agencies and firms often represent a single point of attack and failure. People lack control over the efforts of credit agencies and marketing firms to mine information from financial transactions. Merchants that accept credit and debit cards have a history of security failures, such as the theft of credit card data from Target [16, 25] or the recent security breach of Equifax [22] exposing the personal information of millions.

While VCs offer solutions to some problems posed by centralized entities, they also introduce an additional set of concerns. In this dissertation, we contribute several complementary mechanisms to increase a blockchain system's security, efficiency, and transparency. Our proposed mechanisms work together or are separately deployable, and are applicable to any blockchain based network protocol. Our results can also be used by blockchain designers to understand the consequences of setting the global parameters of their networks.

Contributions

The following is a summary of the contributions in each chapter of this dissertation.

1. **Analysis and improvement of the algorithm for setting difficulty.** We show that the current algorithm used by Bitcoin Core for setting difficulty has bias and high variance, and we derive an alternative estimator based on information available in the blockchain. We show that by using only the inter-arrival time of blocks (if accurate), we can estimate difficulty with no bias and lower variance. However, because many blockchain systems have weak time synchronization requirements allowing for falsification, any third-party's out-of-band record of the time cannot be secured using only information in the blockchain. Therefore, we also propose to create a second estimator based on compact *status reports* regularly broadcast from miners. We plan to examine hybrid approaches that use both estimators, allowing for incremental deployment.

2. **Novel method for status report and block distribution.** We contribute an efficient method fpr propagating blocks and status reports called *Graphene*. We show that our blocks are a fraction of the size of related methods mathematically and via simulation.

3. **Evaluation of the practical feasibility of selfish mining and double spend attacks.** Recent studies of attacks, such as double spending and selfish mining, on blockchain systems fall short because of the simplicity of their model and resulting inability to capture the full complexity of the problem. In particular, there are hidden factors such as the mining power of the network and market fluctuations of the given cryptocurrency that are difficult to estimate for a peer in the network. Therefore, we propose to evaluate these attacks using reinforcement learning, where environmental factors are unknown to an agent.

Collaborators

All research activities are conducted under the supervision of Brian Levine. Preliminary work [23] for Chapter 2 was completed in collaboration with George Bissias and Brian Levine; work [24] for Chapter 3 was completed in collaboration with Gavin Andresen, George Bissias, Amir Houmansadr, and Brian Levine; and work for Chapter 4 was completed in collaboration with Philip Thomas and Brian Levine.

CHAPTER 1

OVERVIEW OF BLOCKCHAIN SYSTEMS

In this chapter, we describe the basic operation of blockchains using Bitcoin Core as an example. Other cryptocurrencies such as Litecoin and Ethereum operate similarly with minor differences.

1.1 Basic Operation

Accounts. A bitcoin is a unit of currency, which is fungible, divisible (up to eight decimal places), and recombinable. It is measured as a balance across multiple accounts, which are themselves manifested in *addresses*.[1] Each address comprises a stored asymmetric cryptographic key and an associated balance of Bitcoin. The public portions of an address are the public key and the balance of coin. When an address is involved in a *transaction* with one or more other addresses, bitcoins are transferred among them. Addresses are explicitly pseudonyms, and not tied to a particular individual; further, empty addresses can be created at no cost beyond generating an asymmetric key pair.

Adding to the blockchain. To be added to the *blockchain*, transactions are broadcast by users on Bitcoin's peer-to-peer (p2p) network. A set of *miners* on the p2p network verify that each transaction is signed correctly, does not conflict with a previous transaction, does not move more coin than is contained in the address, and other functions. Each miner independently agglomerates a set of valid transactions

[1] Internally, Bitcoins exist only as "unspent transaction outputs" (UTXO), but users of the system think of them as balances in addresses.

into a candidate *block* and attempts to solve a predefined cryptographic puzzle as *proof-of-work* (POW), which involves data from the candidate block and a specific *prior block*. The new transactions are only valid if they do not conflict with the set of transactions that are contained in all blocks that are direct ancestors.

The first miner to solve the problem broadcasts his solution to the network, and by virtue of the solution, is able to add the block to the ever-growing blockchain as a child of the prior block. The miners then start over, using the newly appended blockchain and the set of remaining transactions. The miners' incentive for *discovering* a new block is a reward of coins, called the *coinbase*, consisting of a predetermined *block reward* (currently worth 12.5 Bitcoins) and fees from transactions included in the block.

In Bitcoin Core, the POW computation is dynamically calibrated to take approximately ten minutes per block. When transactions appear in a block, they are *confirmed*, and each subsequent block provides additional confirmation. To announce a new block, a miner lists all transactions contained in the new block along with a header that contains an easily-verifiable POW solution. When a node or miner receives a new block, he validates each transaction in the block and the POW.

Notably, if there is a fork on the chain, honest miners always select the prior block as the last block containing the largest amount of POW. However, due to propagation delays in the network, it is possible for the miners to receive competing (but valid) block announcements, which bifurcates the chain, until one of the two forks is appended to first. It is also possible and valid for a miner to receive a set of blocks that retroactively rewrites many blocks; doing so is a demonstration of computational work that miners accept despite the age or depth[2] of a rewritten block.

[2]The *depth* of a block refers to the number of blocks that follow it; the *height* of a block is the number of blocks that precede it.

Any entity can elect to be a miner for Bitcoin Core, and there is no centralized party from whom to seek approval for mining. If all miners were to simply vote on which block should be appended to the main chain, then the mining process would fall vulnerable to a *Sybil attack* [9], where an attacker presents himself as multiple identities on the network. The POW puzzle addresses this problem by performing a kind of decentralized leader election: the miner that solves the puzzle can decide which block to append to the chain.

Full nodes. *Full nodes* are peers in the network that do not mine, but do generate, validate, and propagate transactions and blocks to other nodes including miners. Consumers (i.e., those who purchase goods or services) typically have no need to process and validate all transactions, so they can instead operate *simple payment verification* (SPV) nodes that process, store, and transmit data involving only addresses-of-interest, which are typically addresses they control, make payments to, or receive payments from. SPV nodes rely on full nodes to relay transactions-of-interest.

Bitcoin transaction consistency. The main goal of the p2p network is to provide a consistent view of blocks and unconfirmed transactions across all network peers. Each peer maintains a local snapshot of the transactions in a memory pool dubbed the *mempool*. Blocks consist of a list of transactions that have already (almost always) been broadcast to miners and full nodes in the network.

CHAPTER 2

DIFFICULTY ESTIMATION

Developers have resorted to adhoc methods for updating the difficulty in many blockchain systems. So far, there has been no previous work on analyzing the efficiency and correctness of these methods. In fact, because some blockchain systems do not accurately update difficulty, networks see enormous variance in inter-block delay. If mining power were constant in these networks, then difficulty could be kept constant. However, as seen with many blockchain systems, including a recent example with Bitcoin Cash, mining power fluctuates within these networks, requiring a readjustment of network parameters, including difficulty. Therefore, in this chapter, we use the most prominent cryptocurrency, Bitcoin Core, as a testbed for analyzing difficulty, and then propose alternative methods that could potentially increase efficiency.

2.1 Proof-of-Work in Blockchain Systems

VCs such as Bitcoin Core use a simple POW algorithm based on cryptographic hashing, first proposed by Back [2]. Specifically, miners apply a 256-bit cryptographic hash algorithm [17] to an 80-byte *block header*, and the puzzle is solved if the resulting value is less than a known *target*, $0 < T < 2^{256}$. The header in Bitcoin Core consists of the *Merkle root* [20] of the set of transactions, a timestamp, the target, a *nonce* that is a random number, and the hash of the prior block's header. A Merkle tree is a hash based data structure that is a tree. The tree is constructed such that each leaf is a hash of some piece of data (i.e., the hash of a transaction), and each non-leaf node is a hash of its children. In Bitcoin Core, if the hash of the block header is not less than the target, then a new nonce is selected to generate a new hash (the Merkle root can be adjusted as well). This process repeats until some miner finds a solution. The fraction of attempted hashes a miner performs with respect to the number of hashes performed by the entire network within a given time (i.e., the hash rate) is referred to as the miner's *mining power*.

Each time a nonce is selected and the block header is hashed, the miner is sampling a value from a discrete uniform distribution with range $[0, 2^{256} - 1]$. The probability of solving the POW and discovering a block is the cumulative probability of selecting a value from $[0, T]$, which is $T/2^{256}$. Hence, in expectation, the number of samples needed to discover a block is $2^{256}/T$. Bitcoin Core adjusts the target so that on average it takes about 600 minutes to find a block. Typically, the target is described for convenience as a *difficulty*, defined to be $D = 2^{224}/T$. Bitcoin Core's difficulty is set once every two weeks.

Ethereum. Ethereum operates very similarly to Bitcoin Core. Miners solve a POW problem that is more complicated than Bitcoin Core in an attempt to disadvantage miners with custom ASICs. However, in the end, a miner still compares a hash value to the target. Specifically, the number of values in the block header is larger, resulting in a 508-byte header. It's not the hash of the header that is compared against the target, but the hash resulting from an Ethereum-specific algorithm called ETHASH [11], for which the hash of the block header is the primary input. In the end, the POW hash value is a sample from a discrete uniform distribution with range $[0, 2^{256} - 1]$, and the probability of block discovery is $T/2^{256}$.

A major difference of Ethereum is that the target is set such that the expected time between blocks is 15 seconds. This setting results in quicker confirmation times, but as a result, the probability that two miners announce blocks within the propagation time of a block announcement is much higher. Therefore, there are many abandoned forks in the chain. Ethereum uses a modified version of the GHOST [29] protocol for selecting the main fork of the blockchain: the main chain follows the block at each level with the most POW on its subtree. These differences do not affect the application of our algorithms; in fact, the presence of abandoned forks is additional data which improves our estimates.

2.2 Preliminary Work

In the following section, we describe Bitcoin Core's algorithm for setting difficulty and analyze its statistical properties. Then we describe an alternative estimator and compute the same set of statistical properties for comparison.

2.2.1 Analysis of Bitcoin Core's Difficulty Estimation

Algorithm for setting difficulty. Bitcoin Core's networks parameters are set such that a block is discovered every 10 minutes. Initially, difficulty starts at 1, and then for every 2016 blocks that are found, the timestamps of the blocks are compared to compute the time it took to find 2016 blocks. Let t denote the time in minutes it took to find 2016 blocks. Because the network is configured such that 2016 blocks must take 2 weeks (20160 minutes), the old difficulty is multiplied by $20160/t$. If the correction factor is greater than 4 or less than $1/4$, then 4 or $1/4$ are used, respectively, to prevent the change from being too abrupt.

Bitcoin Core's target and difficulty are related to each other as follows:

$$\text{target} = \frac{\text{targetmax}}{\text{difficulty}} = \frac{2^{224}}{D_i}, \tag{2.1}$$

where D_i is the ith time the difficulty is set [8] and targetmax is the maximum value for the target. This equation shows us that the difficulty and target are inversely proportional: when difficulty increases, the target decreases. Therefore, a smaller target makes block creation more difficult, and as the difficulty goes up, so does the expected time needed to create a block. Next, we use concepts from queuing theory to compute the expected value and variance of Bitcoin Core's difficulty algorithm.

Analysis of difficulty adjustment. Let D_i denote the ith time the difficulty is set, and X_k denote the number of minutes it took to generate the kth block after D_i is set. Mining is an example of a Poisson process because under constant mining power, blocks are mined continuously and independently at a constant average rate. Therefore, $X_k \sim \texttt{Exp}(\beta)$, with $\beta = 1/\lambda$, assuming a fixed hash rate for the 2-week period after D_i is set. In this parametrization of the Exponential, β represents the survival parameter, and hence, the ratio describing the *expected time* it takes for one block to arrive. For example, ideally in Bitcoin Core $\beta = 1/10$ minutes and in Ethereum $\beta = 1/15$ seconds. Using Bitcoin Core's algorithm, the difficulty for the $i+1$st time given D_i and $X_1, \ldots, X_n$, where $n = 2016$, is defined as

$$D_{i+1} = 10nD_i\left(\frac{1}{\sum_{k=1}^{n} X_k}\right). \tag{2.2}$$

Next, we compute the expected value and variance of the difficulty each time it is set by Bitcoin Core's algorithm.

Expected value of difficulty. Let D be a sequence generated by Equation 2.2 where the first element of the sequence is $D_0 = 1$. We can talk about the expected value of a term in sequence D, given its preceding term and the new data we see.

$$\begin{aligned}
\mathbb{E}[D_{i+1}|D_i, X_1, \ldots, X_n] &= \mathbb{E}\left[10nD_i\frac{1}{\sum_{k=1}^{n} X_k}\right] && (2.3)\\
&= 10nD_i\,\mathbb{E}\left[\frac{1}{\sum_{k=1}^{n} X_k}\right] && (2.4)\\
&= 10nD_i\left(\frac{1}{\beta(n-1)}\right) && (2.5)\\
&= \frac{10nD_i}{\beta(n-1)}. && (2.6)
\end{aligned}$$

The sum of any number of exponential random variables is a gamma distribution, and Equation 2.5 can be obtained from Equation 2.4 using the definition of the expected value of an inverse gamma distribution.

Variance of difficulty. Additionally, we can also talk about the variance of a term in sequence D, given its preceding term and the new data we see.

$$\mathrm{Var}(D_{i+1}|D_i, X_1, \ldots, X_n) = \mathrm{Var}\left(10nD_i \frac{1}{\sum_{k=1}^{n} X_k}\right) \tag{2.7}$$

$$= (10nD_i)^2 \mathrm{Var}\left(\frac{1}{\sum_{k=1}^{n} X_k}\right) \tag{2.8}$$

$$= (10nD_i)^2 \left(\frac{1}{\beta^2(n-1)^2(n-2)}\right) \tag{2.9}$$

$$= \frac{(10nD_i)^2}{\beta^2(n-1)^2(n-2)}. \tag{2.10}$$

Once again, the sum of exponential random variables is a gamma distribution, and Equation 2.9 can be obtained from Equation 2.8 using the definition of variance for an inverse gamma distribution.

Next, we compute the bias and mean squared error (MSE) associated with Bitcoin Core's difficulty algorithm. However, in order to compute bias, we need the true underlying difficulty of the network. Therefore, we first explain how to compute the network's hash rate which we use to adjust difficulty correctly. This difficulty is then used for the calculation of bias and MSE, two statistical properties which we will compare with those of an alternative estimator.

The relationship between hash rate and $\boldsymbol{\beta}$. Given difficulty D_i, the expected number of hashes, h, needed to meet the target for a block is

$$\mathbb{E}[h] = \frac{2^{256}-1}{T_i} = \frac{2^{256}-1}{2^{224}/D_i} = \frac{D_i(2^{256}-1)}{2^{224}}, \tag{2.11}$$

where T_i is the target set for the ith time. Note that $\mathbb{E}[h]$ describes the *total* number of expected hashes needed to discover a block, and we have observations regarding the *time* it takes to generate a block. Let r be the hash rate of the network in minutes (i.e., the number of hashes per time unit), and X be a random variable describing the inter-arrival of blocks such that $X = X_1, \ldots, X_n$, where $X \sim \texttt{Exp}(\beta)$, with $\beta = 1/\lambda$. The expected number of hashes needed for a block to be discovered is described by r/β.

$$\mathbb{E}[h] = r\frac{1}{\beta} \tag{2.12}$$

$$r = \mathbb{E}[h]\beta. \tag{2.13}$$

Adjusting difficulty correctly. For Bitcoin Core, where the network is expected to solve a block every 10 minutes, the target for the $i+1$th time is adjusted as follows

$$\frac{(2^{256}-1)}{T_{i+1}} = 10r \tag{2.14}$$

$$\frac{(2^{256}-1)}{T_{i+1}} = 10\,\mathbb{E}[h]\beta \tag{2.15}$$

$$\frac{(2^{256}-1)}{T_{i+1}} = 10\frac{(2^{256}-1)}{T_i}\beta. \tag{2.16}$$

We rearrange Equation 2.16 to solve for T_{i+1}.

$$T_{i+1} = \frac{T_i}{10\beta}. \tag{2.17}$$

Substituting T_i and T_{i+1} with its definition using D_i and D_{i+1}, respectively, we have

$$\frac{2^{224}}{D_{i+1}} = \frac{2^{224}/D_i}{10\beta}. \tag{2.18}$$

Then we solve for difficulty for the $i+1$th time as follows

$$D_{i+1}\frac{2^{224}}{D_i} = 2^{224}10\beta \tag{2.19}$$

$$D_{i+1} = 10\beta D_i. \tag{2.20}$$

Using Equation 2.20, we now can calculate the bias and MSE of Bitcoin Core's algorithm for setting difficulty.

Bias of difficulty.

$$\mathrm{bias}(D_{i+1}|D_i, X_1, ..., X_n) = \mathbb{E}[D_{i+1}|D_i, X_1, ..., X_n] - D_{i+1} \tag{2.21}$$

$$= \frac{10nD_i}{\beta(n-1)} - 10\beta D_i. \tag{2.22}$$

MSE of difficulty.

$$\mathrm{MSE}(D_{i+1}|D_i, X_1, ..., X_n) = \mathrm{bias}(D_{i+1}|D_i, X_1, ..., X_n)^2 + \mathrm{Var}(D_{i+1}|D_i, X_1, ..., X_n) \tag{2.23}$$

$$= \left(\frac{10nD_i}{\beta(n-1)} - 10\beta D_i\right)^2 + \frac{(10nD_i)^2}{\beta^2(n-1)^2(n-2)}. \tag{2.24}$$

2.2.2 Alternative Time Based Estimation

Now we propose an alternative estimator, using the inter-arrival time of blocks, and show that it has lower variance, bias and MSE.

Estimator for $\boldsymbol{\beta}$, the ratio of the expected time between blocks. Let $X = X_1, \ldots, X_n$ denote the inter-arrival time between $n+1$ consecutive blocks on the blockchain. Given a consecutive sequence of $n+1$ blocks, n inter-arrival times can be computed by subtracting the timestamp of each block from that of its preceding block. Note that $X \sim \mathtt{Exp}(\beta)$ with $\beta = 1/\lambda$, similar to the definition in the previous section. It it well known that the unbiased MLE estimator, $\hat{\beta}$, for β is

$$\hat{\beta} = \frac{\sum_{k=1}^{n} X_i}{n}. \tag{2.25}$$

Adjusting difficulty. Using our estimate of β, we can adjust the difficulty for the $i+1$th time as follows

$$D_{i+1} = 10 D_i \hat{\beta}. \tag{2.26}$$

Next, we compute the expected value, variance, bias and MSE associated with this new estimator.

Expected Value of New Difficulty.

$$\begin{aligned}
\mathbb{E}[D_{i+1}|D_i, X_1, \ldots, X_n] &= \mathbb{E}[10 D_i \hat{\beta}] && (2.27)\\
&= 10 D_i \, \mathbb{E}[\hat{\beta}] && (2.28)\\
&= 10 D_i \beta. && (2.29)
\end{aligned}$$

Variance of New Difficulty.

$$\begin{aligned}
\mathrm{Var}(D_{i+1}|D_i, X_1, \ldots, X_n) &= \mathrm{Var}(10 D_i \hat{\beta}) && (2.30)\\
&= (10 D_i)^2 \mathrm{Var}(\hat{\beta}) && (2.31)\\
&= \frac{(10 D_i \beta)^2}{n}. && (2.32)
\end{aligned}$$

Bias of New Difficulty.

$$\begin{aligned}
\mathrm{bias}(D_{i+1}|D_i, X_1, ..., X_n) &= \mathbb{E}[D_{i+1}|D_i, X_1, ..., X_n] - D_{i+1} && (2.33)\\
&= 10 D_i \beta - 10 D_i \beta && (2.34)\\
&= 0. && (2.35)
\end{aligned}$$

MSE of New Difficulty.

$$\mathrm{MSE}(D_{i+1}|D_i, X_1, ..., X_n) = \mathrm{bias}(D_{i+1}|D_i, X_1, ..., X_n)^2 + \mathrm{Var}(D_{i+1}|D_i, X_1, ..., X_n) \tag{2.36}$$

$$= \frac{(10D_i\beta)^2}{n}. \tag{2.37}$$

2.2.3 Comparison of Estimators

Note that our alternative estimator has zero bias compared to Bitcoin Core's original estimator. Additionally, under the appropriate constraints, variance and MSE are also significantly lower.

Variance.

$$\frac{(10D_i\beta)^2}{n} \leq \frac{(10nD_i)^2}{\beta^2(n-1)^2(n-2)} \tag{2.38}$$

$$\frac{\beta^2}{n} \leq \frac{n^2}{\beta^2(n-1)^2(n-2)} \tag{2.39}$$

Our estimator (LHS) has lower variance than the original estimator (RHS) for $n > 2$ and $0 < \beta < 1$.

MSE.

$$\frac{(10D_i\beta)^2}{n} \leq \left(\frac{10nD_i}{\beta(n-1)} - 10\beta D_i\right)^2 + \frac{(10nD_i)^2}{\beta^2(n-1)^2(n-2)} \tag{2.40}$$

Our estimator (LHS) has lower variance than the original estimator (RHS) for $n > 2$, $0 < \beta < 1$ and $D_i \in \mathcal{R}$.

Figure 2.1 and 2.2 represent a summary of our results. Bitcoin Core and Litecoin use 2016 blocks to adjust difficulty, while Ethereum only uses a single block. For both cases, our estimator's variance and MSE are a few orders of magnitude smaller than the current method of estimation.

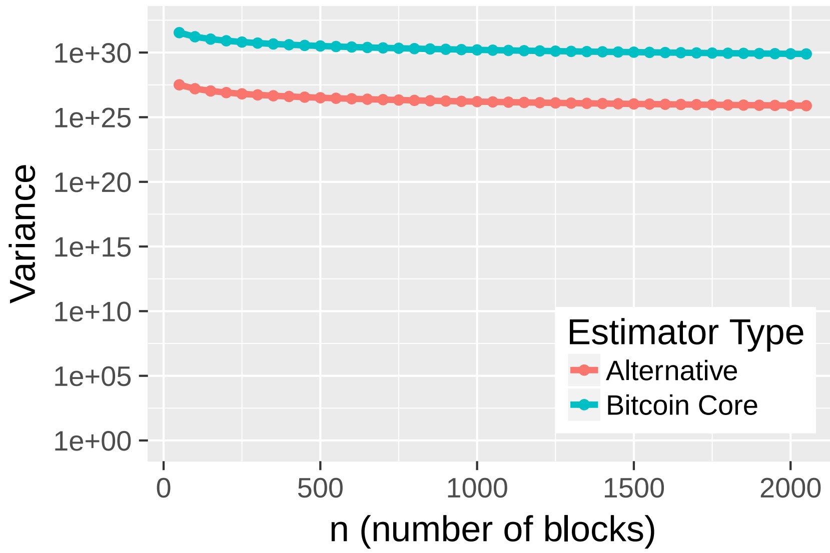

Figure 2.1. Variance of the difficulty algorithm of Bitcoin Core and our estimator, as the number of blocks increases.

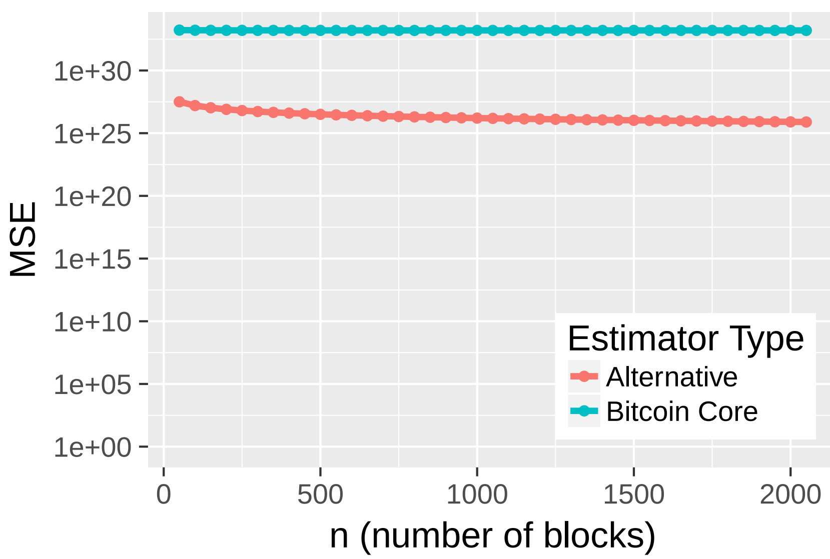

Figure 2.2. Mean squared error of the difficulty algorithm of Bitcoin Core and our estimator, as the number of blocks increases.

2.3 Proposed Work

In this section, we introduce *status reports* that are to be used for a second estimator, and then proposed work relating to this chapter.

2.3.1 Status Reports

Attackers can easily lie about the timestamps on the blocks with our new time based estimator. Therefore, an alternative estimator we plan to develop uses status reports, which is a block header except that the POW does not satisfy the current target. Every α minutes, a parameter determined by the network, the miners send periodic status reports that include the minimum hash value, which represents the hash found since the last block broadcast on the chain. These reports add no computational load to miners, and are stored neither on the blockchain nor at peers that receive them past their usefulness. They are small and can be broadcast out-of-band, for example via RSS or Twitter. Just like block headers, reports are verifiable as authentic POW by third parties. To be clear, each status report does not directly report the minimum hash value; instead, reports are of the input values to the POW algorithm. Because attackers can't lie about their POW, an estimator based on the minimum hash value is safer and can be used to adjust the emergency difficulty when the network fails to produce a block for an unusually long time. We plan to compute the the variance and MSE of the status report based estimator, and compare its performance with other methods.

2.3.2 Key Questions

We were able to show that our time based estimator performs better than Bitcoin Core's unjustified method. To extend this chapter, we propose to answer the following questions centered around a few key themes:

1. **Analysis of attacks against our estimators.** What happens when the timestamps on the blocks are reported inaccurately by an attacker? How much error can an attacker introduce?

2. **Security bounds of the estimators.** What are the Chernoff bounds associated with our estimators?

3. **Settings of network parameters and their consequences.** How often should difficulty be adjusted? How often should miners send status reports? Should non-overlapping windows of blocks be used to compute difficulty at each iteration? Should there be some overlap of blocks between windows at each iteration?

4. **Individual parameter settings for peers.** Given a required error rate, what is the shortest window of time (or number of blocks) needed to estimate difficulty correctly?

5. **Fluctuations in mining power.** Given that mining power is changing, how quickly can either estimator adapt to change?

6. **Pros and cons of our estimators.** What are the advantages and disadvantages of status reports compared to the time based estimator? For example, the status report based estimator uses the minimum hash value, which is arguably safer because miners have to show their POW. On the other hand, for the time based estimator, there is no POW required when reporting timestamps.

CHAPTER 3

GRAPHENE: EFFICIENT BLOCK ANNOUNCEMENTS

In this chapter, we explain Graphene, a novel protocol for the propagation of blocks and status reports. The main limitation we are addressing with Graphene is the inefficiency of blockchain systems in propagating block data. A block announcement must be validated using the transaction content comprising the block. However, it is likely that the majority of the peers have already received these transactions, and they only need to discern them from those in their mempool. Additionally, in Section 2.3, we propose using status reports (i.e. blocks announcements that do not satisfy the difficulty requirement of the network) for the emergency difficulty algorithm. Therefore, on top standard blocks, we are proposing to add more network traffic, requiring an even more efficient method of block propagation.

3.1 Background

In this subsection, we describe the signaling mechanism behind blockchain systems, explain the operation of Invertible Bloom Lookup Tables and summarize related work.

3.1.1 Topology and Signaling in Blockchain Systems

Bitcoin Core propagates new transaction and block announcements by flooding throughout a p2p random graph of full nodes and miners. Each peer in the graph requests direct connections to 8 other peers, and accepts requests for connections from up to 117 other peers. A peer will offer a newly created transaction to each neighbor via an `inv` message, which reports the hash of the transaction content as its ID. If a peer does not already possess the transaction, it will request it using a `getdata` message. Blocks are handled similarly: `inv` messages describe a block by its ID, which is created from the hash of the block's contents. Upon receiving the `inv`, peers will request the block if they do not already have it. Hence, in today's topology, `inv` messages cross every edge in the random graph once, while the actual transaction and block data typically propagate along only a spanning tree of the graph (more edges will be traversed if there are propagation delays). For convenience, we refer to the set of (unconfirmed) transaction IDs that a peer knows about as the *IDpool*. Actual transaction contents are placed in the mempool.

3.1.2 Related Work

In principle, a block announcement needs to include only the IDs of those transactions, and accordingly, Corallo's *Compact Block* design [7] — which has been recently deployed — significantly reduces block size by including a transaction ID list at the cost of increasing coordination to 3 roundtrip times. *Xtreme Thinblocks* [31], an alternative protocol, works similarly to Compact Blocks but has greater data overhead. Specifically, Xtreme Thinblocks utilize a compact data structure known as a Bloom filter, that allow two parties to determine, with high probability, which values from a set they share in common. A Bloom filter is an array of x bits representing y items. Initially, the x bits are cleared. Whenever an item is added to the filter, k bits, selected using k hash functions, in the bit-array are set. The number of bits required by the filter is

$$x = \frac{-y\ln(f)}{\ln^2(2)}, \tag{3.1}$$

where f is the intended false positive rate (FPR). The protocol is as follows. After receiving an `inv` for a block, the receiver creates a Bloom filter of her mempool with FPR $f = 1/n$, where n is the number of transactions in the block. The sender then sends a *thinblock transaction* that contains block header information, all transaction IDs in the block and any transactions that do not pass through the Bloom filter, enabling the receiver to recreate the block. As a result, Xtreme Thinblocks are larger than Compact Blocks but require just 2 roundtrip times. Relatedly, the community has discussed in forums the use of IBLTs (alone) for reducing block announcements [1, 26], but these schemes have not been formally evaluated and are less efficient than our approach. Our novel method, which we prove and demonstrate is smaller than all of these recent works, requires just 2 roundtrip times for coordination.

3.1.3 Overview of Invertible Bloom Lookup Tables

An IBLT [14] is an efficient data structure for *set reconciliation*, where the goal is for two parties, each holding a set of items, to obtain the union of the two sets. Like Bloom filters [5], IBLTs allow two parties to determine the intersection of the values from a set they mostly share in common. But unlike Bloom filters, IBLTs additionally enable the recovery of any missing values, which are assumed to be of fixed size and encoded as binary strings. Elements in a set can be inserted, retrieved and deleted like an ordinary hash table. We now describe the operation of IBLTs relevant to our protocol.

For our purposes, an IBLT consists of m entries, each storing a `count`, a `hashSum` and a `valueSum`, all initialized to zero. A new value v is inserted into location $i = h(v)$ based on the hash of its value such that $i < m$. At entry i, all three fields are incremented or `xor`'ed. In particular, standard addition is used for the `count` field, but `xor` is used to add to the `hashSum` and `valueSum` fields. An item can be deleted similarly: at the correct entry, `count` is subtracted by 1, and the `hashSum` and `valueSum` fields are `xor`'ed. When `count` $= 1$ the `valueSum` field contains the actual value of the sole item remaining in the cell. (The purpose of the `keySum` field is to support a `GET`() operation for a given key: that is, if `count` $= 1$ and `hashSum` $= h(v)$, then `valueSum` $= v$.) IBLTs use $k > 1$ hash functions to store each value in k entries, which we collectively call a value's *entry set*. If table space is sufficient, then with high probability for at least one of the k entries, `count` $= 1$.

Suppose that two peers each have a list of values, V and V', respectively, such that the difference is expected to be small. The first peer constructs an IBLT L (with m entries) from V. The second peer constructs V' from L' (also having m entries). Eppstein et al. [10] showed that a cell-by-cell difference operator can be used to efficiently compute the symmetric difference $L \triangle L'$. For each pair of fields (f, f'), at each entry in L and L', we compute either $f \oplus f'$ or $f - f'$ depending on the field type. When $|$`count`$| = 1$ at any entry, the corresponding value can be recovered. Peers proceed by removing the recoverable items from all entries in the value's entry set. This process will generally produce new recoverable entries, and continues until nothing is recoverable.

3.2 The Protocol

In this section, we detail Graphene, where a receiver learns the set of specific transaction IDs that are contained in a (pending or confirmed) block containing n transactions. Unlike other approaches, Graphene never sends an explicit list of transaction IDs, instead it sends a small Bloom filter and a very small IBLT.

PROTOCOL 1: Graphene

1: Sender: Sends `inv` for a block.

2: Receiver: Requests unknown block; includes count of transactions in her IDpool, m.

3: Sender: Sends Bloom filter $\mathcal{S}$ and IBLT $\mathcal{I}$ (each created from the set of n txn IDs in the block) and essential Bitcoin header fields. The FPR of the filter is $f = a/(m-n)$, where $a = n/(c\tau)$.

4: Receiver: Creates IBLT $\mathcal{I}'$ from the txn IDs that pass through $\mathcal{S}$. She decodes the *subtraction* [10] of the two blocks, $\mathcal{I} \triangle \mathcal{I}'$.

The intuition behind Graphene is as follows. The sender creates an IBLT $\mathcal{I}$ from the set of transaction IDs in the block. To help the receiver create the same (or similar) IBLT, he also creates a Bloom filter $\mathcal{S}$ of the transaction IDs in the block. The receiver uses $\mathcal{S}$ to filter out transaction IDs from her IDpool and creates her own IBLT $\mathcal{I}'$. She then attempts to use $\mathcal{I}'$ to *decode* $\mathcal{I}$, which, if successful, will yield the transaction IDs comprising the block. The number of transactions that falsely appear to be in $\mathcal{S}$, and therefore are wrongly added to $\mathcal{I}'$, is determined by a parameter controlled by the sender. Using this parameter, he can create $\mathcal{I}$ such that it will decode with very high probability.

For Graphene, we set $f = a/(m-n)$, where a is the expected difference between $\mathcal{I}$ and $\mathcal{I}'$. Since the Bloom filter contains n entries, and we need to convert to bytes, its size is

$$\frac{-\ln(\frac{a}{m-n})}{\ln^2(2)}\frac{1}{8}.$$

It is also the case that a is the primary parameter of the IBLT size. IBLT $\mathcal{I}$ can be decoded by IBLT $\mathcal{I}'$ with very high probability if the number of cells in $\mathcal{I}$ is d-times the expected symmetric difference between the list of entries in $\mathcal{I}$ and the list of entries in $\mathcal{I}'$. In our case, the expected difference is a, and we set $d = 1.5$ (see Eppstein et al. [10], which explores settings of d). Each cell in an IBLT has a `count`, `hashSum` and `valueSum`. (It can also have a key, but we have no need for a key). For us, the count field is 2 bytes, the hashSum is 4 bytes, and the valueSum is the last 6 bytes of the transaction ID (which is sufficient to prevent collisions). In sum, the size of the IBLT with a symmetric difference of a entries is $1.5(2+4+6)a = 18a$ bytes. Thus, the total cost in bytes, T, for the Bloom filter and IBLT are given by

$$T(a) = n\frac{-\ln(f)}{c} + a\tau = n\frac{-\ln(\frac{a}{m-\mu})}{c} + a\tau,$$

where all Bloom filter constants are grouped together as $c = 8\ln^2(2)$, and we let the overhead on IBLT entries be the constant $\tau = 18$.

To set the Bloom filter as small as possible, we must ensure that the FPR of the filter is as high as permitted. If we assume that all `inv` messages are sent ahead of a block, we know that the receiver already has all of the transactions in the block in her IDpool (they need not be in her mempool). Thus, $\mu = n$; i.e., we allow for a of $m - n$ transactions to become false positives, since all transactions in the block are already guaranteed to pass through the filter. It follows that

$$T(a) = n\frac{-\ln(\frac{a}{m-n})}{c} + a\tau. \tag{3.2}$$

Taking the derivative w.r.t. a, Eq. 3.2 is minimized[1] when when $a = n/(c\tau)$.

Due to the randomized nature of an IBLT, there is a non-zero chance that it will fail to decode. In that case, the sender resends the IBLT with double the number of cells (which is still very small). In our simulations, presented in the next section, this doubling was sufficient for the incredibly few IBLTs that failed.

PROTOCOL 2: CompactBlocks

1: Sender: Sends `inv` for a block that has n transactions.

2: Receiver: If block is not in mempool, requests a compact block.

3: Sender: Sends the block header information, all transaction IDs in the block and any full transactions he predicts the sender hasn't received yet.

[1] Actual implementations of Bloom filters and IBLTs involve several (non-continuous) ceiling functions such that we can re-write:

$$T(a) = \left(\lceil \ln(\frac{m-n}{a}) \rceil \left\lceil \frac{n\ln(\frac{m-n}{a})}{\lceil \ln(\frac{m-n}{a}) \rceil \ln^2(2)} \right\rceil\right) \frac{1}{8} + \lceil a \rceil \tau. \tag{3.3}$$

The optimal value of Eq. 3.3 can be found with a simple brute force loop. We compared the value of a picked by using $a = n/(c\tau)$ to the cost for that a from Eq. 3.3, for valid combinations of $50 \leq n \leq 2000$ and $50 \leq m \leq 10000$. We found that it is always within 37% of the cost of the optimal value from Eq. 3.3, with a median difference of 16%. In practice, a for-loop brute-force search for the lowest value of a is almost no cost to perform, and we do so in our simulations.

4: Receiver: Recreates the block and requests missing transactions if there exist any.

3.3 Comparison to Compact Blocks

In this section, we mathematically show the efficiency of Graphene to Compact Blocks and present our simulation results comparing the two protocols.

3.3.1 Mathematical Analysis

Compact Blocks [7] is to our knowledge the best-performing related work. It has several modes of operation, and we examined the *Low Bandwidth Relaying* mode due to its bandwidth efficiency, which operates as follows. After fully validating a new block, the sender sends an `inv`, for which the receiver sends a `getdata` message if she doesn't have the block. The sender then sends a Compact Block that contains block header information, all transaction IDs (shortened to 6 bytes) in the block, and any transactions that he predicts the receiver does not have (e.g., the coinbase). If the receiver still has missing transactions, she requests them via an `inv` message. Protocol 2 outlines this mode of Compact Blocks. The main difference between Graphene and Compact Blocks is that instead of sending a Bloom filter and an IBLT, the sender sends block header information and all shortened transaction IDs to the receiver.

A detailed example of how to calculate the size of each scheme is below; but we can state more generally the following result. For a block of n transactions, Compact Blocks costs $6n$ bytes. For both protocols, the receiver needs the `inv` messages for the set of transactions in the block before the sender can send it. Therefore, we expect the size of the IDpool of the receiver, m, to be constrained such that $m \geq n$. Assuming that $m > 0$ and $n > 0$, the following inequality must hold for Graphene to outperform Compact Blocks:

$$
\begin{aligned}
n\frac{-\ln(\frac{a}{m-n})}{c} + a\tau &< 6n \qquad (3.4)\\
n &> \frac{m}{55116364}. \qquad (3.5)
\end{aligned}
$$

In other words, Graphene is strictly more efficient than Compact Blocks *unless* the set of unconfirmed transactions held by peers is $55,116,364$ times larger than the block size (e.g., over 110 billion unconfirmed transactions for the current block size.) Finally, we note that Xtreme Thinblocks [31] are strictly larger than Compact Blocks since they contain all IDs and a Bloom filter, and therefore Graphene performs strictly better than Xtreme Thinblocks as well.

Example. A receiver with an IDpool of $m = 4000$ transactions makes a request for a new block that has $n = 2000$ transactions. The value of a that minimizes the cost is $a = n/(c\tau) = 28.9$. The sender creates a Bloom filter $\mathcal{S}$ with $f = a/(m - n) = 28.9/2000 = 0.01445$, with total size of $2000 \times -ln(0.01445)/c = 2.2$ KB. The sender also creates an IBLT with a cells, totaling $18a = 520B$. In sum, a total of $2204B + 520B = 2.7$ KB bytes are sent. The receiver creates an IBLT of the same size, and using the technique introduced in Eppstein et al. [10], the receiver subtracts one IBLT from the other before decoding. In comparison, for a block of n transactions, Compact Blocks costs $2000 \times 6B = 12$ KB, over 4 times the cost of Graphene.

Ordered blocks. Graphene does not specify an order for transactions in the blocks, and instead assumes that transactions are sorted by ID. Bitcoin Core requires transactions depending on another transaction in the same block to appear later, but a canonical ordering is easy to specify. If a miner would like to order transactions with some proprietary method (e.g., [15]), that ordering would be sent alongside the IBLT. For a block of n items, in the worst case, the list will be $n\log_2(n)$ bits long. Even with this extra data, our approach is much more efficient than Compact Blocks. In terms of the example above, if Graphene was to impose an ordering, the additional cost for

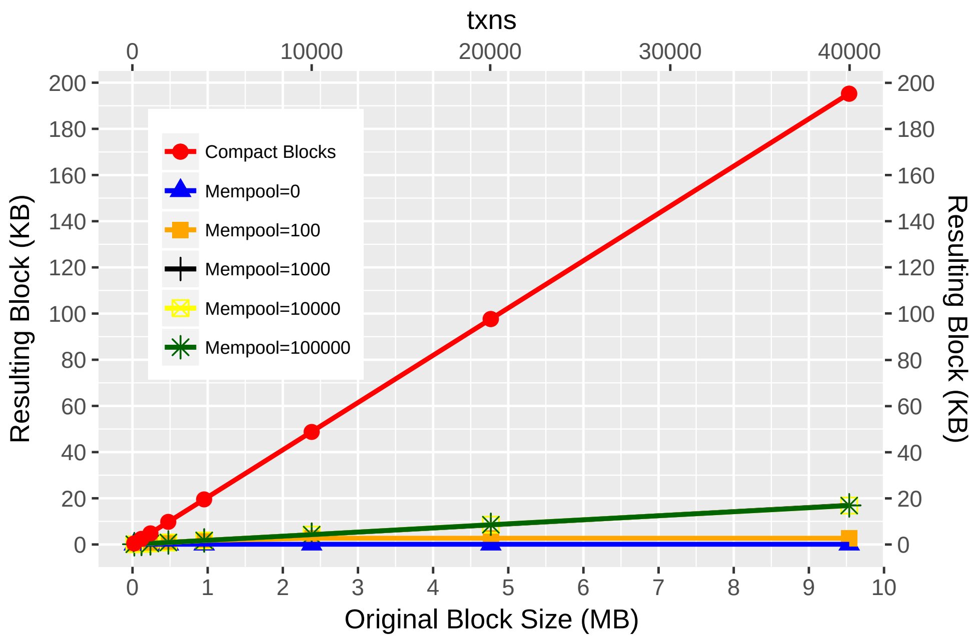

Figure 3.1. A comparison of Graphene to Compact Blocks. Mempools are expressed in MB and the corresponding number of transactions. Graphene is at most 1/10th the size of Compact Blocks.

$n = 2000$ transactions would be $n \log_2(n)$ bits $= 2000 \times log_2(2000)$ bits $= 2.74$ KB. This increases the cost of Graphene to 5.44 KB, still less than half of Compact Blocks.

3.3.2 Empirical Evaluation

Simulation of Bloom filters and IBLTs. Figure 2 shows the results of a simulation of Compact Blocks and Graphene. The simulation is comprised of many trials, where each trial takes as input a block size (in terms of transactions) and the mempool size. Each protocol is executed and the number of bytes required is recorded. Because Bloom filters and IBLTs are probabilistic mechanisms, the simulation uses the real data structures to ensure the accuracy of the results. The plot is the mean of hundreds of simulations at that point, and error bars are too small to be shown. As the figure shows, Graphene is consistently 1/10 of the cost of Compact Blocks or less, depending on the mempool size. As mempool size increases, the growth of Graphene blocks to Compact Blocks is extremely slow.

CHAPTER 4

REINFORCEMENT LEARNING APPLIED TO BLOCKCHAIN SYSTEMS

The mining power of miners is the primary quantitative factor that determines the security of any POW based blockchain consensus algorithm. Recent models [12,13,27] on attacks against blockchain systems assume that the mining power of honest and malicious miners is known by the attacker. However, estimation of mining power in blockchain systems introduces error into these models. In this chapter, we quantify the error associated with estimating the mining power of miners on Bitcoin Core, and justify our reasoning for using Reinforcement Learning (RL) methods to more accurately quantify the security of blockchain systems.

4.1 Background

In this section, we describe double spend and selfish mining attacks on blockchain systems, and explain the paradigms used to analyze these attacks.

4.1.1 Double Spending

In a *double spend* attack [21], an attacker creates a transaction that moves funds to a merchant's address. After the transaction appears in the newest block on the main branch, the attacker takes possession of the purchased goods. Using his mining power, the attacker then immediately releases two blocks, with a transaction in the first that moves the funds to a second attacker-owned address. Now the attacker has the goods and his coin back. To defend against the attack, a merchant can refuse to release goods to a customer until z blocks have been added to the blockchain, including the first block containing a transaction moving coin to the merchant's address. Nakamoto calculated the probability of the attack succeeding assuming that the miner controlled a given fraction of the mining power [21]; for a given fraction, the probability of success decreases exponentially as z increases.

In general, a merchant may wait z blocks before releasing goods, which can thwart an attacker. But choosing the minimum value of z that secures a transaction is an unresolved issue. The core Bitcoin client shows that a transaction is unconfirmed until it is 6 blocks deep in the blockchain [4], and advice from others is necessarily vague; e.g., "for very large transactions, coin owners might want to wait for a larger number of block confirmations" [6].

Recently, Gervais et al. [13] evaluated the security of blockchains in terms of an attacker's economic profitability and assuming finite resources. They modeled a double spending attacker's strategy as a Markov Decision Process (MDP) [3]. MDPs are defined by a finite set of discrete states, a set of actions, a transition function, and a reward function. They defined each state in the MDP as a tuple representing the following: the status of the fork, the number of blocks mined by an attacker, the number of blocks mined by the honest miners, and the number of blocks mined by an *eclipse attack* victim [18], respectively. In an eclipse attack, an attacker controls all outgoing connections of some set of peers, thereby preventing them from receiving fresh block and transaction data. Gervais et al. encoded several factors into the MDP that affect attacker strategy, including mining power, block depth, connectivity, and the impact of eclipse attacks. The MDP was implemented with a cutoff value of 20 blocks, representing an attack of finite duration. Using a search algorithm over the space defined by the MDP that simulates a blockchain system, they determined the maximum transaction value that would be safe from double spending by an economically rational attacker. This approach is rich, capturing the optimal strategy for double spending (as well as *selfish mining* [12, 27]) given network conditions and blockchain parameters. Gervais et al. were able to reach interesting conclusions about the comparative performance and security of several widely used blockchains.

Sapirshtein et al. [27] first observed that some double spend attacks can be carried out essentially cost-free in the presence of concurrent selfish mining [12] attacks. More recent work extends the scope of double spends that can benefit from selfish mining to cases where the attacker is capable of *pre-mining* blocks on a secret branch at little or no opportunity cost [30]. The papers identify the optimal mining strategy for an attacker and quantify the advantage he can expect to have over the merchant in terms of pre-mined blocks.

4.1.2 Selfish Mining

The standard Bitcoin Core protocol requires miners to broadcast a block they mined immediately. However, in the case of *selfish mining*, a miner deliberately withholds transaction information. The motivation is to bifurcate the chain and waste the computational resources of the honest miners should the network decide to build on the attacker's chain. An attacker can't profit economically since the number of blocks that can be created by a miner depends on the fraction of mining power he has. However, selfish mining discards the honest miners' blocks, by releasing an alternative chain that takes over the current longest chain. If a selfish mining attack is successful, the attacker owns a higher fraction of the blocks on the main chain because some portion of the blocks created by the honest network go to waste.

Eyal et al. [12] inroduced a selfish mining strategy and analyzed it using a Markov chain [19]. Markov chains are defined by set of discrete states and a transition function that describes the probability of moving from a given state to another. Then Sapirshtein et al. [27] created a more complex model using an MDP for selfish mining and computed the ϵ-optimal policy that increases a selfish miner's revenue (i.e., the fraction of blocks owned by the selfish miner on the main chain). Recently, Gervais et al. [13] incorporated additional network parameters into the MDP to study their affects on the attacker's policy.

4.2 Estimation of Mining Power

Bitcoin Core does not have an algorithm for estimating a given miner's mining power. Therefore, we derive a simple method, using an analysis similar to that in Chapter 2.

Let $X^m = X_1^m, \ldots, X_n^m$, where $X^m \sim \text{Exp}(\beta_m)$, with $\beta_m = 1/\lambda_m$, denote n inter-arrival times of $n+1$ blocks discovered by a *single miner*. Additionally, let $X = X_1, \ldots, X_n$, where $X \sim \text{Exp}(\beta)$, with $\beta = 1/\lambda$, denote n inter-arrival times of the most recent $n+1$ blocks discovered by *all miners*. In Section 2.2.2, we have shown that the unbiased MLE estimator for any β is equation 2.25 that is the sample mean. Hence, can use this estimator to calculate the hash rate of a given miner and the entire network. It follows that the miner's mining power, ψ_m, is the entire network's hash rate over his own hash rate:

$$\psi_m = \frac{\hat{\beta}}{\hat{\beta}_m}. \tag{4.1}$$

Simulation. For a miner with a given mining power and n, we wrote a simulation that created a blockchain including inter-arrival times, and estimated $\hat{\beta}$ and $\hat{\beta}_m$. Then using Equation 4.1, we computed the miner's mining power. By repeating the experiment over many trials, we created an empirical distribution for our estimate of the miner's mining power. For each mining power, we sampled from our empirical distribution to be fed to the MDP of Gervais [13] that outputs an optimal strategy for the attacker. We evaluated this strategy by running it over many trials, computing the fraction of blocks created by the attacker on the main chain. Figure 4.1 summarizes our results as we increase the attacker's mining power. The y-axis represents the

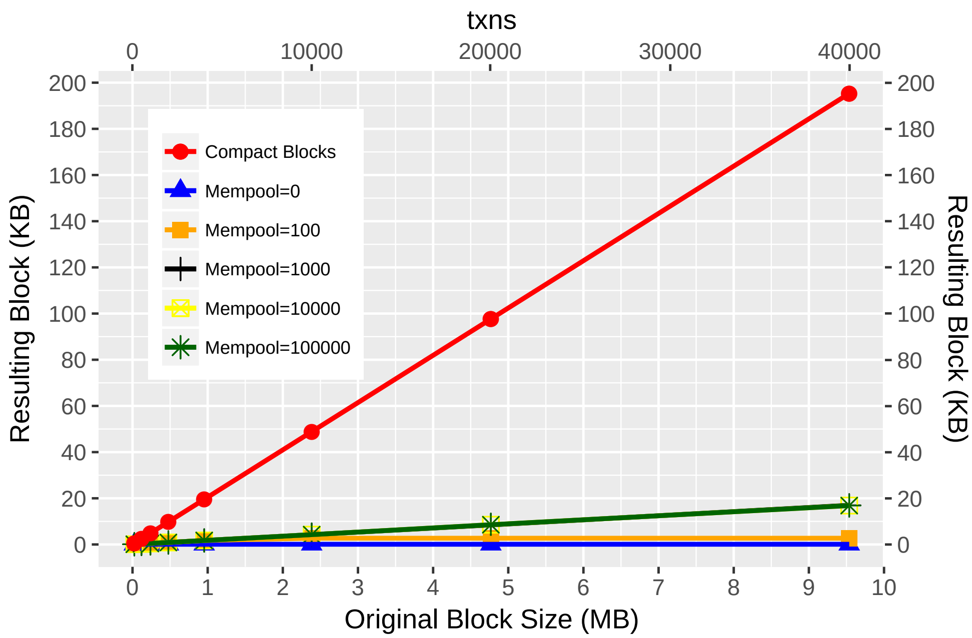

Figure 4.1.

fraction of blocks created by the attacker on the main chain. The line marked as 'Gervais' and that marked as 'Eyal' represent the results from the original MDP and Markov chain, respectively. The line marked as 'Altered', uses our empirical distribution on the model of Gervais [13].

4.3 Preliminary Work

Sapirshtein [27] and Gervais [13] use an *average reward* MDP for computing the optimal strategies for selfish mining and double spending. Average reward MDPs, as the name suggests, maximize an agent's average return instead of *episodic* MDPs that maximize absolute return over time. In the following section, we formulate the problem as an episodic MDP that is more widely studied and easier to solve.

4.3.1 Formulating the Problem as an Episodic MDP

States. Using the model model of Sapirshtein et al. [27] for selfish mining as a basis, we construct the following MDP that is a 6-tuple $\{S, A, P, R, \gamma, d_0\}$, where $S = \{(w, x, y, z, i, k)\}$ such that $w, x, y, z, i, k \in \mathbb{N}$. The state consists of a 6-tuple where each element represents the following: 1) w is the number of blocks created by the honest miners on the main chain. 2) x is the number of blocks created by the attacker. 3) y is the number of attacker blocks that the honest network accepts as part of the main chain. 4) z is a variable that represents the state of the main chain. If $z = 1$, the attacker performed a `match` action, resulting in a fork on the main chain. If $z = 0$, the attacker mined the last block, and if $z = 2$, the honest network mined the last block, enabling the attacker to release a competing block if he has any. 5) i is the number of attacker blocks on the main chain so far. 6) k is the *total* number of blocks on the main chain so far. Note that this representation assumes that all blocks build on the same parent block.

Actions. $A = \{\texttt{adopt}, \texttt{mine}, \texttt{override}, \texttt{match}\}$. `adopt` refers to the adoption of the main chain, thereby discarding all blocks created by the attacker (except those already accepted on the main chain by the honest network). The action `mine` denotes that the attacker continues to mine, waiting to see who the next block will be discovered by. `override` refers to an attacker's releasing one more block than the honest miners' blocks on the main chain. This action can be viewed as honest or selfish depending on the current state. If the honest miners have no blocks on the main chain, an addition of a block to the main chain means that the attacker is honest. However, if the honest miners already have blocks on the main chain and the attacker releases an alternative chain that is 1 block longer than that created by the honest miners, then the attacker overwrites the main chain, wasting the victim's computational resources. The `match` action means that the attacker releases as many blocks as there are on the main chain, causing a bifurcation.

Initial state distribution. $d_0 = \{(0,0,0,0,0,0)\}$, where $P(S_0 = (0,0,0,0,0,0)) = 1$. In other words, the start state assumes that no blocks have been mined yet. If the attacker chooses to `adopt` and the length of the main chain is greater than some CUTOFF, the agent goes back to the start state.

Transition Function. We consider 3 of parameters of interest also included in the model of Gervais et al. [13]: q, CUTOFF and a. At each time step, a new block is created by the network: with probability q, where q is the mining power of the attacker, the attacker is the winner of a new block. The honest network discovers a block with probability $1-q$. Not all actions are available in every state. The attacker can always choose to `mine` or `adopt`. If the attacker chooses an `adopt` action, he discards all blocks he has created on his alternative chain and accepts the blocks on the main chain created by the honest network should there exist any. Furthermore, `override` and `match` are only available when the attacker has enough blocks and the last block has been mined by the honest network. At CUTOFF = 75, the length of the main chain is equal to or longer than 75 blocks, and we force the attacker to `adopt` in order to have episodic trials. Our third parameter, a, represents network connectivity. If there is a fork of same length on the main chain, fraction a of the honest miners build on the attacker's alternative chain. We set $a = 1$ to give advantage to the attacker, and to analyze if RL methods can learn to take advantage of the `match` action. Therefore, in our formulation, if the attacker matches the main chain with a fork of the same length, all honest miners build on the attacker's chain.

4.4 Proposed Work

We propose to complete the following tasks to extend this chapter:

1. Prove that the episodic MDP presented above is equivalent to the average reward MDP of Sapirshtein et al. [27].

2. Evaluate the model of Sapirshtein et al. [27] where an agent first calculates the hash rate of the attacker using our estimator.

3. Run RL algorithms such as Q-learning and SARSA on the MDP presented, where network parameters are not revealed to an agent.

4. Compare the performance of RL algorithms to previous work.

5. Evaluate RL algorithms on the model of Sapirshtein et al. [27] when the mining power of the network is fluctuating.

CHAPTER 5

TIMELINE

- May 2018: Dissertation proposal
- June-August 2018: Complete proposed work in Section 2.3 regarding difficulty estimation in blockchain systems
- September-October 2018: Revise work on difficulty estimation for paper submission
- November 2018-January 2019: Complete proposed work in Section 4.4 regarding applying RL algorithms to blockchain systems
- February-March 2019: Revise work on RL algorithms for paper submission
- April-May 2019: Update workshop paper on Chapter 3 regarding Graphene for full paper submission
- May 2019: Dissertation defense

BIBLIOGRAPHY

[1] Andresen, Gavin. O(1) Block Propagation. https://gist.github.com/gavinandresen/e20c3b5a1d4b97f79ac2, August 2014.

[2] Back, Adam. Hashcash - Amortizable Publicly Auditable Cost-Functions, 2002.

[3] Bellman, Richard. A markovian decision process. *Journal of Mathematics and Mechanics* (1957), 679–684.

[4] Confirmation. https://en.bitcoin.it/wiki/Confirmation, February 2015.

[5] Bloom, Burton H. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM 13*, 7 (July 1970), 422–426.

[6] Bonneau, Joseph. How long does it take for a bitcoin transaction to be confirmed? https://coincenter.org/2015/11/what-does-it-mean-for-a-bitcoin-transaction-to-be-confirmed/, November 2015.

[7] Corallo, Matt. Bip152: Compact block relay. https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki, April 2016.

[8] Difficulty. http://learnmeabitcoin.com/guide/difficulty, Feb 2015.

[9] Douceur, J. The Sybil Attack. In *Proc. Intl Wkshp on Peer-to-Peer Systems (IPTPS)* (Mar. 2002).

[10] Eppstein, David, Goodrich, Michael T., Uyeda, Frank, and Varghese, George. What's the Difference?: Efficient Set Reconciliation Without Prior Context. In *ACM SIGCOMM* (2011).

[11] Ethash. https://github.com/ethereum/wiki/wiki/Ethash, Last retrieved June 2017.

[12] Eyal, Ittay, and Sirer, Emin Gün. Majority is not enough: Bitcoin mining is vulnerable. *Financial Cryptography* (2014), 436–454.

[13] Gervais, Arthur, O. Karame, Ghassan, Wust, Karl, Glykantzis, Vasileios, Ritzdorf, Hubert, and Capkun, Srdjan. On the Security and Performance of Proof of Work Blockchains. https://eprint.iacr.org/2016/555, 2016.

[14] Goodrich, M.T., and Mitzenmacher, M. Invertible bloom lookup tables. In *Conf. on Comm., Control, and Computing* (Sept 2011), pp. 792–799.

[15] Hanke, Timo. A Speedup for Bitcoin Mining. `http://arxiv.org/pdf/1604.00575.pdf` (Rev. 5), March 31 2016.

[16] Harris, Elizabeth A., and Perlroth, Nicole. For Target, the Breach Numbers Grow. *The New York Times* (Jan. 11 2014), B1.

[17] Hashcash. `https://en.bitcoin.it/wiki/Hashcash`, Last retrieved June 2017.

[18] Heilman, Ethan, Kendler, Alison, Zohar, Aviv, and Goldberg, Sharon. Eclipse Attacks on Bitcoin's Peer-to-peer Network. In *USENIX Security* (2015).

[19] Markov, Andrey. Extension of the limit theorems of probability theory to a sum of variables connected in a chain.

[20] Merkle, Ralph C. A digital signature based on a conventional encryption function. In *Conference on the Theory and Application of Cryptographic Techniques* (1987), Springer, pp. 369–378.

[21] Nakamoto, Satoshi. Bitcoin: A Peer-to-Peer Electronic Cash System, May 2009.

[22] Newman, Lily Hay. Security news this week: Equifax found 2.4 million more people affected by its 2017 breach. `https://www.wired.com/story/equifax-found-millions-more-people-affected-2017-breach/`.

[23] Ozisik, A. Pinar, Andresen, Gavin, Bissias, George, Houmansadr, Amir, and Levine, Brian Neil. Graphene: A New Protocol for Block Propagation Using Set Reconciliation. In *Proc. of International Workshop on Cryptocurrencies and Blockchain Technology (ESORICS Workshop)* (September 2017).

[24] Ozisik, A. Pinar, Bissias, George, and Levine, Brian N. Estimation of miner hash rates and consensus on blockchains. Tech. rep., PDF available from arxiv.org and `https://www.cs.umass.edu/~brian/status-reports.pdf`, June 2017.

[25] Perlroth, Nicole. Target struck in the cat-and-mouse game of credit theft. *The New York Times* (20 December 2013), B1.

[26] Russel, Rusty. Playing with invertible bloom lookup tables and bitcoin transactions. `http://rustyrussell.github.io/pettycoin/2014/11/05/Playing-with-invertible-bloom-lookup-tables-and-bitcoin-transactions.html`, Nov 2014.

[27] Sapirshtein, Ayelet, Sompolinsky, Yonatan, and Zohar, Aviv. Optimal Selfish Mining Strategies in Bitcoin. `https://arxiv.org/pdf/1507.06183.pdf`, July 2015.

[28] Sasson, Eli Ben, Chiesa, Alessandro, Garman, Christina, Green, Matthew, Miers, Ian, Tromer, Eran, and Virza, Madars. Zerocash: Decentralized anonymous payments from bitcoin. In *IEEE S&P* (2014), pp. 459–474.

[29] Sompolinsky, Yonatan, and Zohar, Aviv. Secure high-rate transaction processing in Bitcoin. *Financial Cryptography and Data Security* (2015).

[30] Sompolinsky, Yonatan, and Zohar, Aviv. Bitcoin's Security Model Revisited. `https://arxiv.org/abs/1605.09193`, May 2016.

[31] Tschipper, Peter. BUIP010 Xtreme Thinblocks. `https://bitco.in/forum/threads/buip010-passed-xtreme-thinblocks.774/`, Jan 2016.