QUANTIFYING AND IMPROVING THE SECURITY 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

September 2019

College of Information and Computer Sciences

QUANTIFYING AND IMPROVING THE SECURITY 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 SECURITY OF BLOCKCHAIN SYSTEMS

SEPTEMBER 2019

A. PINAR OZISIK

B.Sc., BRANDEIS UNIVERSITY

M.S., UNIVERSITY OF MASSACHUSETTS

Directed by: Professor Brian N. Levine

In this thesis, I analyze and improve the security and performance of blockchain systems across three primary themes. In the first theme, I analyze blockchain algorithms for setting block discovery difficulty. Unfortunately, churn 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 has seen enormous variance in mining power since its creation and the existing algorithm for difficulty did not easily converge. I propose two alternatives to accurately update difficulty: one that solely uses information that is currently available in blockchain networks, and another based on status reports regularly broadcast from some or all miners of their partial proof-of-work (POW). Status reports can also be used for emergency difficulty adjustment, an algorithm the network resorts to when a block takes unusually long to discover.

Status reports add overhead into networks because they require the broadcast of additional information. In a second theme, I introduce a novel method of interactive set reconciliation for the distribution of status reports in order to reduce traffic. Even without status reports, this protocol works for the efficient distribution of blocks. The approach, called Graphene, couples a Bloom filter with an IBLT. Then I evaluate performance analytically and show that Graphene blocks are always smaller and therefore network performance is improved.

In the third theme, I analyze the practical feasibility of double-spend and selfish mining attacks on blockchain systems. The hash rate of miners is the primary quantitative factor that determines the security of any POW based blockchain consensus algorithm. Most analyses generally assume that the hash rate of honest and malicious miners is known. However, I show that hash rate estimation is difficult and introduces high variance. Therefore, I argue that these double-spend and selfish mining attacks are difficult to carry out with high precision, and use reinforcement learning techniques to realistically evaluate these attacks when an attacker does not have full knowledge of the networks mining power.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

INTRODUCTION

Contributions

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

1.

Collaborators

CHAPTER 1

OVERVIEW OF BLOCKCHAIN SYSTEMS

1.1 Basic Operation

Account balances. 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.

Roles. Users wishing to exchange coins broadcast the details of their transactions over Bitcoin's p2p network, signed with their private keys. A set of *miners* on the p2p network verify that each transaction is signed correctly and does not conflict with another transaction. Miners independently agglomerate a set of valid transactions into a *block* and attempt to solve a predefined proof-of-work (POW) problem involving this block and a chain of prior valid blocks. In Bitcoin, the POW computation is dynamically calibrated to take approximately ten minutes per block. The first miner to solve the problem broadcasts his solution to the network, adding it to the ever-growing *blockchain*; the miners then start over, with the appended blockchain and the set of transactions that were not added as part of the previous block. When transactions appear in a block, they are considered *confirmed*, and each subsequent

[1] Internally, Bitcoins exist only as "unspent transaction outputs" (UTXO), but users of the system think of them as balances in addresses, and that view does not affect the results of this paper.

block provides additional confirmation. The miners' incentive for discovering a block is a reward of coins, called the *coinbase*, consisting of a predetermined *block reward* (currently 12.5 BTC) and fees from transactions included in the block.

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 Bitcoin 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.

To announce a new block, a miner lists all transactions contained in the new block along with a header that provides an easily verifiable *proof-of-work* (POW) solution. When a full node or miner receives a new block, it validates each transaction in the block and the proof of work.

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.

Topology and flooding. Bitcoin 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, in this paper, 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.

CHAPTER 2

DIFFICULTY ESTIMATION

2.1 Problem Statement

2.2 Preliminary Work

2.2.1 Analysis of Difficulty Estimation

2.2.2 Alternative Estimation

2.3 Proposed Work

CHAPTER 3

GRAPHENE

3.1 Background

In this section, we review the operation of IBLTs and summarize related work.

3.1.1 Overview of IBLTs

Overview of IBLTs. We make use of Invertible Bloom Lookup Tables (IBLTs) [5], which is an efficient data structure for *set reconciliation* between two peers. Like Bloom filters [2], IBLTs allow two parties to determine, with high probability, which values from a set they share in common. But unlike Bloom filters, IBLTs enable the recovery of any missing values, which are assumed to be of fixed size and encoded as binary strings. Key-value pairs can be inserted, retrieved and deleted like an ordinary hash table. An IBLT consists of m entries, each storing a `count`, a `keySum`, 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 xored. 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` $\equiv 1$.

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. [4] 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 $|\texttt{count}| \equiv 1$ at any entry, the corresponding value can be recovered. Peers proceed by removing the recoverable key-value pair from all entries in the value's entry set. This process will generally produce new recoverable entries, and continues until nothing is recoverable.

3.1.2 Related Work

The main limitation we are addressing with Graphene is the inefficiency of blockchain systems in disseminating 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. In principle, a block announcement needs to include only the IDs of those transactions, and accordingly, Corallo's *Compact Block* design [3] — 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. We further detail Compact Block's operation in Section **??** and compare it quantitatively in Section **??**. *Xtreme Thinblocks* [8], an alternative protocol, works similarly to Compact Blocks but has greater data overhead. Specifically, if an `inv`is sent for a block that is not in the receiver's mempool, the receiver sends a Bloom filter of her IDpool along with the request for the missing 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, 7], 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.

3.2 Graphene: Efficient Block Announcements

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 txns 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 $f = \frac{a}{m-n}$, where $a = n/(c\tau)$.

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

3.2.1 The Protocol

The intuition behind Graphene is as follows. The sender creates an IBLT $\mathcal{I}$from the set of transaction (txn) IDs in the block. To help the receiver create the same IBLT (or similar), 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 pool of received transaction IDs (which we call the 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.

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 $x =$

$y\frac{-\ln(f)}{\ln^2(2)}$, where f is the intended false positive rate (FPR). For Graphene, we set $f = \frac{a}{m-n}$, 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. [4], which explores settings of d). Each cell in an IBLT has a *count*, a *hash* value, and a stored *value*. (It can also have a key, but we have no need for a key). For us, the count field is 2 bytes, the hash value is 4 bytes, and the value is the last 5 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 $1.5(2 + 4 + 5)a = 16.5a$ 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 $\tau = 16.5$.

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

$$T(a) = n\frac{-\ln(\frac{a}{m-n})}{c} + a\tau. \tag{3.1}$$

Taking the derivative w.r.t. a, Eq. 3.1 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 txns.

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

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

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

3.2.2 Comparison to Compact Blocks.

Compact Blocks [3] is to our knowledge the best-performing related work. It has several modes of operation. 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 5 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.

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

$$T(a) = \left(\lceil \ln(\frac{m-n}{a}) \rceil \left\lceil \frac{n \ln(\frac{m-n}{a})}{\lceil \ln(\frac{m-n}{a}) \rceil \ln^2(2)} \right\rceil \right) \frac{1}{8} + \lceil a \rceil \tau. \tag{3.2}$$

The optimal value of Eq. 3.2 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. 3.2, 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. 3.2, 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.

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 $5n$ 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:

$$n\frac{-\ln(\frac{a}{m-n})}{c} + a\tau < 5n \tag{3.3}$$

$$n > {}^{m}/_{1287670} \tag{3.4}$$

In other words, Graphene is strictly more efficient than Compact Blocks *unless* the set of unconfirmed transactions held by peers is 1,287,670 times larger than the block size (e.g., over 22 billion unconfirmed transactions for the current block size.) Finally, we note that Xtreme Thinblocks [8] 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. In Section **??**, we provide specific empirical results from network simulation, where we use real IBLTs and Bloom filters to evaluate Graphene and Compact Blocks.

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 $a = n/(c\tau) = 31.5$. The sender creates a Bloom filter $\mathcal{S}$with $f = \frac{a}{m-n} = 31.5/2000 = 0.01577$, with total size of $2000 \times \frac{-ln(0.01577)}{c} = 2.1$ KB. The sender also creates an IBLT

with a cells, totaling $16.5a = 521B$. In sum, a total of $2160B + 521B = 2.6$ KB bytes are sent. The receiver creates an IBLT of the same size, and using the technique introduced in Eppstein et al. [4], the receiver subtracts one IBLT from the other before decoding. In comparison, for a block of n transactions, Compact Blocks costs $2000 \times 5B = 10$ KB, over 3 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 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., [6]), 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 $n = 2000$ transactions would be $n \log_2(n)$ bits $= 2000 \times log_2(2000)$ bits $= 2.74$ KB. This increases the cost of Graphene to 5.34 KB, still almost half of Compact Blocks.

3.2.3 Empirical Evaluation

CHAPTER 4

RL APPLIED TO BITCOIN

CHAPTER 5

TIMELINE

BIBLIOGRAPHY

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

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

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

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

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

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

[7] 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.

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