On the Viability of Distributed Consensus by Proof of Space

Vivek Bhupatiraju John Kuszmaul Vinjai Vale

March 3, 2017

Abstract

In this paper, we present our implementation of Proof of Space (PoS) and our study of its viability in distributed consensus. PoS is a new alternative to the commonly used Proof of Work, which is a protocol at the heart of distributed consensus systems such as Bitcoin. PoS resolves the two major drawbacks of Proof of Work: high energy cost and bias towards individuals with specialized hardware. In PoS, users must store large "hard-to-pebble" PTC graphs, which are recursively generated using subgraphs called superconcentrators. We implemented two types of superconcentrators to examine their differences in performance. Linear superconcentrators are about 1.8 times slower than butterfly superconcentrators, but provide a better lower bound on space consumption. Finally, we discuss our simulation of using PoS to reach consensus in a peer-to-peer network. We conclude that Proof of Space is indeed viable for distributed consensus.

To the best of our knowledge, we are the first to implement linear superconcentrators and to simulate the use of PoS to reach consensus on a decentralized network.

1 Introduction and Motivation

Proof of Space [2] is a new and promising cryptographic primitive. One of its most important potential applications lies in distributed consensus on a decentralized network, which is essential for cryptocurrencies such as Bitcoin. This paper demonstrates experimentally that Proof of Space is effective in this context, and provides the first performance analysis for its use to reach consensus.

Proof of Space was created to improve on its predecessor Proof of Work [4], which originated more than twenty years ago as a technique to increase the difficulty of Distributed Denial of Service (DDoS) attacks on websites and spamming of emails. The purpose of Proof of Work is to allow a prover P to demonstrate to a verifier V that P has spent a significant amount of computational resources on solving some problem. Proofs of Work can be required to increase the difficulty of performing certain tasks to dissuade attackers. For example, if ten seconds of computational time were required to send an email, then spammers could not send millions of emails in a day.

Proof of Work is traditionally implemented using cryptographic hashes. A cryptographic hash is a function that accepts a string as input and returns as output a long, seemingly random string of fixed length. For example, SHA-512("1234") returns the 512 bit (64 byte) string:

d404559f602eab6fd602ac7680dacbfaadd13630335e951f097af3900e9de176b6db285
12f2e000b9d04fba5133e8b1c6e8df59db3a8ab9d60be4b97cc9e81db

Using a cryptographic hash function HASH(), a Proof of Work can be designed as follows. The verifier V sends to P a string r along with a non-negative integer n, which dictates the computational complexity of the Proof of Work. The prover P is tasked with finding a string e such that HASH$(r||e)$ (where || represents concatenation) begins with n zeroes. Since cryptographic hash functions are non-invertible, P must perform an expected 2^n hashes to solve this Proof of Work. After P sends his solution e to V, V can verify the solution by checking that HASH$(r||e)$ begins with at least n zeroes, which requires only a single cryptographic hash computation. Ease of verification is critical to the practicality of Proof of Work. For example, if Google implemented the Proof of Work requirement

to send emails to Gmail users, then Google would have to verify tens of billions of Proofs of Work every day [11], which is only practical if each verification is very fast.

In recent years, one of the most important applications of Proof of Work has become distributed consensus protocol for blockchains on cryptocurrencies such as Bitcoin. The blockchain acts as a public ledger detailing transfers of Bitcoins between users. Bitcoin users use Proof of Work to reach consensus on the blockchain. In particular, Proof of Work is used to allow community members to make contributions to the blockchain without any single community member having the ability to dominate. Because a large amount of work has been used to create the blockchain, users can be fairly certain that it is accurate. The probability of a block's accuracy also increases with its age.

Distributed consensus along with other applications of Proof of Work have brought to light a number of major weaknesses of traditional Proof of Work implementations. Against modern attackers, traditional implementations of Proof of Work are vulnerable to the usage of highly specialized hardware, which may be able to compute cryptographic hashes a hundred times faster, and ten thousand times more energy-efficiently than standard machines [12]. What the verifier may believe to be a ten-second Proof of Work may actually take an attacker only a tenth of a second. Moreover, current implementations of Proof of Work consume large amounts of energy on standard machines, posing a problem for the large scale usage of cryptocurrencies such as Bitcoin [9]. Another major vulnerability of Proof of Work lies in the context of Botnets, which are groups of infected computers that can be utilized by hackers for email spamming or DDoS attacks.

One recently proposed solution to these issues is Proof of Space. Whereas Proof of Work requires computational effort, Proof of Space requires space consumption. Proof of Space allows a prover P to demonstrate to a verifier V that P has devoted a certain amount of space (in memory or in persistent storage) to completing a task. Again, V should be able to verify this Proof of Space easily (devoting few computational resources and little space).

Proof of Space addresses all of the major vulnerabilities of Proof of Work. Proof of Space puts users with different hardware on a more level playing field. Although customized hardware

exists to optimize for cryptographic hashes, memory and persistent storage cannot easily be further optimized for Proof of Space. As was recently demonstrated in SpaceMint [9], Proof of Space can use several orders of magnitude less energy than Proof of Work. Unlike Proof of Work, hackers cannot covertly complete Proofs of Space on botnet machines without ever using a significant amount of resources at once, making it easier for infected users to detect that their device has been compromised. Proof of Space can also replace Proof of Work in preventing DDoS attacks and email spamming.

This paper studies the performance characteristics of Proof of Space in the context of distributed consensus. We discuss our implementation of a full working prover and verifier interface for Proof of Space, as well as our simulated consensus protocol by Proof of Space on a peer-to-peer network.

Because verification for Proof of Space may be slower than for Proof of Work, one concern is that Proof of Space would be slow to reach consensus. Our first main contribution is the simulation of consensus protocol with PoS. We find that after 41 seconds, 90% of 150 users in a network reached consensus. This suggests that Proof of Space is an effective tool for distributed consensus, as our simulation for Proof of Space is only around two times slower than the equivalent simulation for Proof of Work, and both take time well under the ten minutes Bitcoin takes to reach consensus [8].

Proof of Space involves the prover storing large PTC graphs [10], which are recursively defined with superconcentrators. The second main contribution of this paper is to provide experimental data on the differences between using butterfly and linear superconcentrators. Running Proofs of Space on a solid state drive, we find that, in practice, the use of linear superconcentrators is only around 1.8 times slower than the use of butterfly superconcentrators in terms of pebbling time. Interestingly, this suggests that linear superconcentrators are more practical than butterfly superconcentrators for Proof of Space. In particular, the theoretical lower bounds for space consumption with linear superconcentrators is $\Omega\left(\frac{n}{\log n}\right)$, while the lower bound for space consumption with butterfly superconcentrators is $\Omega\left(\frac{n}{\log^2 n}\right)$ [12]. Therefore, the multiplicative difference between the

lower bound for space consumption and the actual performance is significantly better for linear superconcentrators for essentially all sizes of Proof of Space.

Prior to our work, relatively little has been done in the direction of experimental evaluation of Proof of Space. Most notably, the recent paper SpaceMint [9] created a complete prover and verifier environment for Proof of Space, using PTC graphs recursively defined with butterfly superconcentrators. One key difference between our work and their work is that we investigate both butterfly superconcentrators [5] and linear superconcentrators [13]. We also implemented a simulation of distributed consensus on a network and experimentally analyzed the time to consensus by Proof of Space. To the best of our knowledge, we are the first to implement linear superconcentrators, and the first to simulate consensus on a peer-to-peer network for Proof of Space.

The rest of this paper proceeds as follows. In Section 2, we describe the protocols used in Proof of Space, distributed consensus, and the graphs used for implementations of Proof of Space. In Section 3, we discuss our implementations, focusing on the difference between linear and butterfly superconcentrators and our simulation of consensus on networks. In Section 4, we present experimental data for pebbling time, Merkle tree generation time, and time to reach consensus. Finally, in Section 5, we provide conclusions and discuss future work.

2 Background and Technical Discussion

In this section, we begin by discussing distributed consensus and blockchain, which are used in cryptocurrencies such as Bitcoin. We then consider the details of a complete PoS implementation.

2.1 Distributed Consensus and the Bitcoin Blockchain

Distributed consensus is the problem of getting a majority of the users in a network to agree on some information. In the case of Bitcoin, this information is a financial transaction between two individuals; call them Alice and Bob, and consider a transaction where Alice owes Bob twenty

Bitcoins. Both entities need assurance that the transaction will be honored. One potential solution is for Charlie, a middleman, to facilitate the transaction. Then a future dispute could be settled by examining his records. However, Charlie may not be trustworthy or safe from hacking.

Hence the main insight behind blockchain and distributed consensus is trusting the network. Specifically, the only way to manipulate transaction history should require one to overcome the entire honest subset of the network – a scope which is practically impossible for any attack.

Now that we have motivated distributed consensus, let us discuss its specifics. Distributed consensus in Bitcoin is based on a public ledger system known as blockchain. More formally, a blockchain (Figure 1) is a series of blocks, each consisting of four components: a list of transactions to add the the ledger, a Proof of Work, the hash of the previous block (thus creating the chain), and other metadata (the timestamp, identification of the user who created the block, etc.). Every user in the network (also known as a miner) is allowed to submit a new block. A monetary incentive is also provided to the first miner to generate the next block in a chain. Because of the Proof of Work component, formulating a valid block is difficult. Additionally, all the miners in the network are configured to only recognize to the longest existing chain of blocks, ignoring all forks and satellite chains (Figure 2). This makes the Bitcoin blockchain highly tamper-resistant. Suppose an attacker wished to rewrite the transaction history in an old block; this would require not only creating an alternate version of that old block, but also creating enough blocks following that to catch up to the main chain. It is virtually impossible for an attacker to muster enough computational power for this feat, because they need to be able to beat the entire honest portion of the network.

However, as discussed in the introduction this blockchain suffers from energy cost and bias towards specialized hardware. Hence we seek to implement a similar blockchain system, except basing it on Proof of Space instead of Proof of Work.

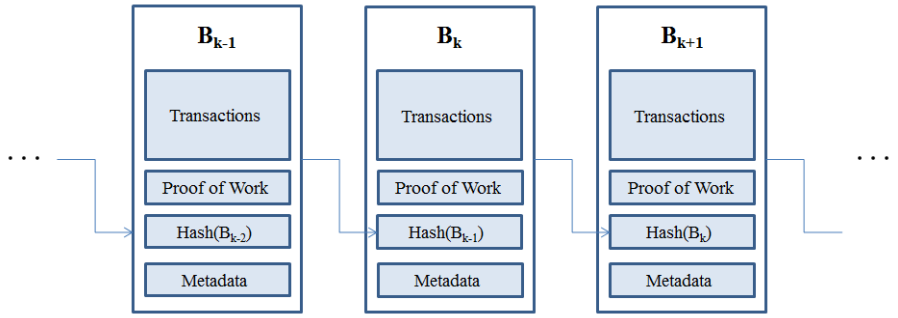

Figure 1: A schematic of the Bitcoin blockchain.

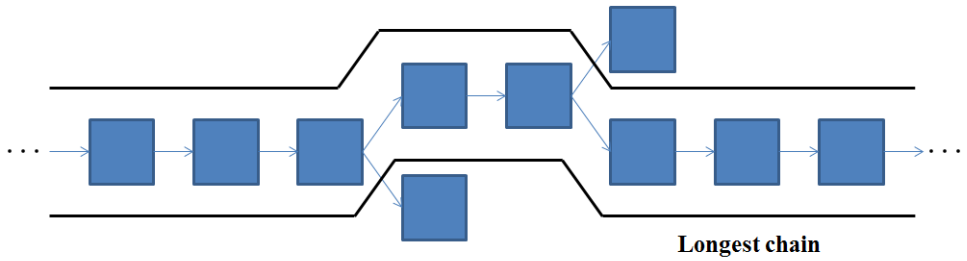

Figure 2: A schematic of the longest-chain protocol.

2.2 Requirements of a PoS

We now consider what we require of a PoS. A PoS must enable the prover P to demonstrate to the verifier V that she has devoted a given and presumably large amount of storage to the PoS. There are two main requirements of such a scheme: the communication between P and V should be small, and V should be able to verify P's proof with small space and computational resources.

A trivial PoS protocol can be conducted as follows. V first generates a random binary string. This binary string should be long enough that it takes a significant amount of space to store. V then sends the binary string to P, to be stored for a given amount of time (say one week). V then randomly chooses several bits in the string; we will use 100 for our example. He remembers those bits and their locations, and then frees the storage he used to hold the rest of the binary string. At the end of the week V asks the prover for the values of those 100 bits. If P is able to correctly answer these queries, V can be confident with high probability that P stored most of the file.

This design fails to meet the first of the two requirements. Sending the large binary string over the Internet is extremely impractical. To address this issue, we will look to a more complex PoS that is based on having P store so-called "hard-to-pebble" graphs.

2.3 The Pebbling Game and Hard-to-Pebble Graphs

We first define the abstract notion of the pebbling game based on [10], and then we proceed to its application to the problem at hand as described in [2]. The Pebbling Game is a single-player game played on a directed acyclic graph with pebbles as tokens. The rules of the game are as follows:

- A pebble may be placed on a vertex if and only if all of its parents have a pebble on them. Hence a source of the graph (a vertex with no parents) may be pebbled at any time.
- A pebble may be removed from a vertex at any time.

The goal of the game is to pebble all the sinks of the graph (the vertices with no children) starting from an unpebbled graph. In particular, one wishes to do this using as few pebbles as possible.

In the context of Proof of Space, this game translates into the following. First, the prover will need to store a large directed acyclic graph. Placing a pebble on a vertex corresponds to computing and storing a value for that vertex. Each vertex's value is calculated by hashing the values of its parents. Thus computing and storing the value of a vertex corresponds to placing a pebble on that vertex. This requires the values of the parents and hence can only be done after the parents are pebbled[1]. Removing a pebble corresponds to freeing that vertex's value from memory or disk space, and that space can be used to store another vertex's value later on. Hence the minimal number of pebbles needed to pebble a sink corresponds directly to the minimal amount of space needed.

Using this setup, one can design a simple PoS in which the prover is tasked with providing the

[1] Additionally, the prover must use their own public key as a seed value for pebbling the sources. During verification, the verifier will ask for some of the source values and check that they came from the prover's public key, in order to confirm the identity of the PoS's creator.

hash values for some of the sink nodes in the graph. This solves the problem of short communication, because there is an established protocol for P to build the graph.

However, we must carefully choose the graph so as to prevent the dishonest prover from only pebbling a portion of the graph (and attempting to then pebble the queried positions during verification). In order to make it difficult for the prover to cheat, we want our graph to be hard to pebble. Intuitively, a graph is hard to pebble if in order to pebble a vertex, one must pebble a large number of its ancestors.

Let us consider what factors influence whether a graph is hard to pebble. Larger graphs are more difficult to pebble, simply because there are more vertices. Also, having more edges per vertex intuitively makes the graph harder to pebble. If the graph is well connected, and no part of it is isolated from others, a prover will have to pebble most of the ancestors of a vertex before pebbling the vertex itself.

In [10], Paul, Tarjan, and Celoni defined a recursive family of graphs, known as the PTC graph. The nth PTC graph PTC(n) is generated by attaching a set of sources, a superconcentrator, two copies of PTC$(n-1)$, another superconcentrator, and finally a set of sinks. (See Figure 3 for an illustration.) A superconcentrator (SC) is defined as a graph such that if one picks any set of sources and a set of sinks both of size n, one can find n vertex-disjoint paths connecting these sources and sinks. More detail is provided about the PTC graph in [10] and [5].

The authors of the PTC paper also prove that there is no pebbling strategy of their graph which uses fewer than $\Omega\left(\frac{n}{\log n}\right)$ pebbles, where n is the total number of vertices in the PTC graph. This result implies that a PoS for a PTC graph of n vertices proves that P has stored at least $\Omega\left(\frac{n}{\log n}\right)$ space, which is a significant amount.

2.4 Efficient Verification with Merkle Trees

In this subsection, we consider an additional step that allows V to efficiently verify P's PoS. Clearly V cannot efficiently pebble the PTC graph himself, so he needs some other way of confirmation.

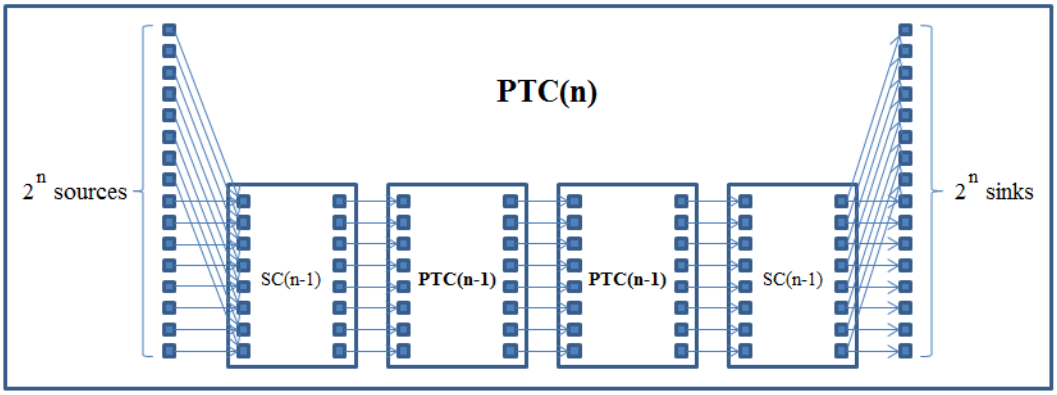

Figure 3: A schematic of the PTC graph.

Merkle trees – a data structure patented by Ralph Merkle in 1979 [7] – provide a working solution [2]. P must build a Merkle tree using the pebbled graph's vertex hash values as the leaves, by doing as follows. First P initializes a binary tree whose leaves are the PTC graph's nodes. As she builds the tree towards the root, she hashes the values of the parents of each node to find the value of the current node. Eventually P will reach the topmost vertex, called the Merkle root. Note that in the context of pebbling games, this is placing a tree on top of our initial graph and pebbling the tree to reach a single sink.

Verification is then performed as follows. First, P broadcasts his Merkle root, which becomes public knowledge. V then challenges P for the hash values of certain randomly selected nodes in the original PTC graph (i.e., leaves in the Merkle tree). Every time P sends over a leaf, she also sends its sibling and the siblings of its descendants in the Merkle tree (called the sibling path). This collective response from P is called the *opening* of the node in question. Upon receiving each opening, V hashes his way up the sibling path until he confirms that the final hash matches the publicly known Merkle root. If the computed Merkle root does not match the Merkle root that P initially broadcasted, V will know that P was not honest and will reject P's PoS. Otherwise, V can be confident that P was storing the entire Merkle tree in order to provide each proof.

V must also challenge P to open some of the sources, in order to confirm that the PoS is really from P and is not stolen from another prover. Recall that P should use her public key as a nonce

value for the sources; V knows P's public key, so he simply rehashes those sources to confirm P's ownership of the PoS.

This method has been proven to work with a small number of challenges in [2]; [9] and [2] recommend 30 challenges as a sufficient number, after the Merkle root has been verified. Note that provers store the same Merkle tree throughout multiple consensus steps. The first time they use this Merkle tree, they are subject to a more extensive suite of verification challenges, in order to verify with high probability that their Merkle root is genuine and that the leaves of their Merkle tree are the pebbled PTC vertices, as claimed; this works by opening more nodes with their parents (in the PTC graph), in order to verify that parents' hash values correctly combine to form the child's hash values in the PTC graph [2].

2.5 PoS Based Blockchain

It is vital that all the users in a blockchain network have a fair shot at passing a block. In a Proof of Work based blockchain, whoever can compute a valid proof first gets to pass their block; this works out because hashing is effectively random. However, for PoS everyone will take approximately the same amount of time to formulate their proof, since the difficulty lies in dedicated a large amount of storage, not generating the PoS. Hence, time cannot be used to determine the "winner" – whichever miner gets their proof chosen for the next block. We require that each user's probability of winning is proportional to the amount of space they dedicate. Hence we must have a standardized quality function performed on every submitted proof, as proposed in [9].

Here we make the simplifying assumption that every user is dedicating the same amount of space. Otherwise our quality function would have to include that parameter. Instead, our quality function simply hashes the entire proof. The valid proof with the smallest hash (based on lexicographical order) is chosen as the winner.

Therefore the full protocol is as follows: when a new miner joins the network, they will receive instructions to generate and store large PTC graphs. Then each time a new block is passed, the

miner will apply a known protocol to extract the next challenge based on the previous block. The miner will generate a PoS based on the challenge and the graph they have stored using their public key as a seed, compile all the new transactions they are aware of into a block, and broadcast their block to the network with their public key attached to it. Additionally, when the miner hears about another block being broadcasted, they will compare it to the current block they are broadcasting. If the new block contains a valid PoS of higher quality, the miner will start broadcasting that block instead, thereby allowing the new block to propagate further into the network.

A more complete description including various theoretical attacks and defenses is provided in [9]. This paper focuses primarily on performance analysis of the PoS.

3 Contributions

Previous implementations of Proof of Space have not used linear superconcentrators in their PTC constructions and have not tested reaching consensus with PoS in a network [9]. We implement both in an attempt to address these two gaps and to better explore the viability of Proof of Space.

3.1 Linear Superconcentrators in PTC Graphs

As discussed in Section 2.3, the main family of graphs used in the Proof of Space protocol are PTC graphs, which involve superconcentrators in their construction. Superconcentrators (SCs) are graphs with the property that if one picks a set of sources and a set of sinks, both of size n, one can find n-vertex disjoint paths connecting these sinks and sources.

There exist multiple families of SCs. Previous implementations of the PoS protocol with PTC graphs [9] have used the family of butterfly superconcentrators (BSCs). A BSC is recursively constructed, with its base case being the complete bipartite graph $K_{2,2}$, denoted BSC(1). Each successive superconcentrator BSC(i) is constructed by connecting a set of 2^i sources and sinks to two BSC$(i-1)$ graphs, with each source of BSC(i) connected to one source of each BSC$(i-1)$

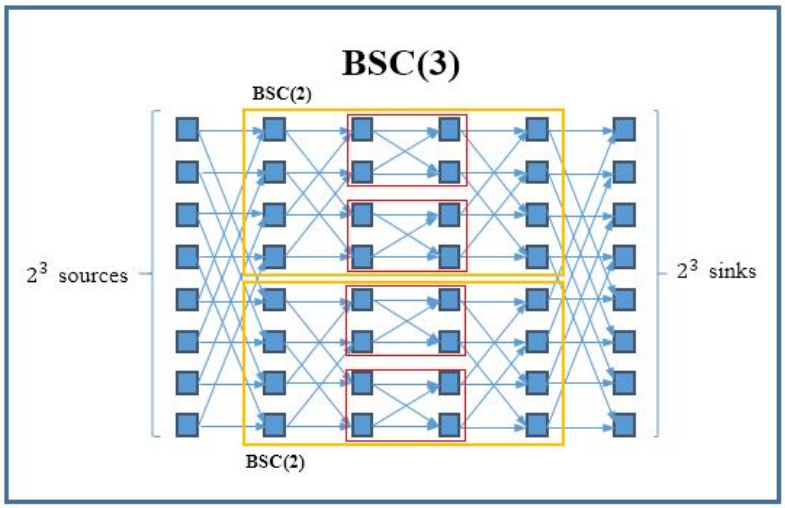

Figure 4: BSC(3), the 3rd Butterfly Superconcentrator.

graph [5]. Additionally, each sink of BSC$(i-1)$ is connected to two sinks of BSC(i). BSC(3) is illustrated in Figure 4, with each BSC(2) graph highlighted. The number of vertices in BSC(i) is $\Theta(i \cdot 2^i)$, meaning the number grows by roughly a factor of 2 on each iteration.

As is also discussed in Section 2.3, Paul, Tarjan, and Celoni have proven that there exists no pebbling strategy of their graph which uses fewer than $\Omega\left(\frac{n}{\log n}\right)$ pebbles, where n is the total number of vertices in the PTC graph [10]. In terms of our protocol, having to pebble $\Omega\left(\frac{n}{\log n}\right)$ vertices is equivalent to computing and storing at least $\Omega\left(\frac{n}{\log n}\right)$ values of vertices, which is a significant amount of space. However, this result assumes that the SC is linear in the number of edges. Hence, the BSC construction used by other PoS implementations does not achieve the $\Omega\left(\frac{n}{\log n}\right)$ bound in the paper; instead, it only achieves a bound of $\Omega\left(\frac{n}{\log^2 n}\right)$ [12].

A tighter bound is more desirable for a PoS setup, as it ensures that the prover has used more space to pebble the graph. As such, we have constructed PTC graphs with linear superconcentrators (LSCs) in order to achieve the PTC bound and to compare the LSC's performance against BSCs.

The LSC we used was also recursively generated, and involves a special bipartite graph called an expander. The expander graph has the same number of left and right vertices, as well as some

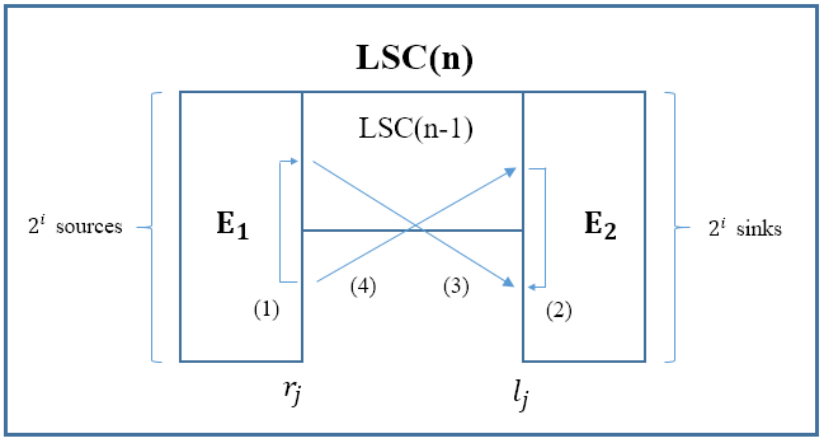

Figure 5: LSC(n), the n^{th} Linear Superconcentrator.

other properties defined in [13]. The LSC's base case is also the complete bipartite graph $K_{2,2}$, denoted LSC(1). Each successive graph LSC(i) consists of one LSC$(i-1)$ graph sandwiched between two expander graphs named E_1 and E_2 with 2^i left and right vertices each. The upper 2^{i-1} right vertices of E_1 are connected with the sources of LSC$(i-1)$, while the upper 2^{i-1} left vertices of E_2 are identified with the sinks of LSC$(i-1)$.

In addition, there are several other edges in the LSC. Denote the right nodes of E_1 as r_j and the left nodes of E_2 as l_j, for $j = 1, 2, \ldots, 2^i$. Then we have the following 2^{i+1} edges: (1) $r_{2^{i-1}+j} \to r_j$, (2) $l_j \to l_{2^{i-1}+j}$, (3) $r_j \to l_{2^{i-1}+j}$, (4) $r_{2^{i-1}+j} \to l_j$, where $j = 1, 2, \ldots, 2^{i-1}$. These connections are illustrated in Figure 5.

It is proven in [13] that this construction satisfies the SC condition and is linear, meaning that if it is used in the PTC construction, the $\Omega\left(\frac{n}{\log n}\right)$ bound proven in [10] will hold.

To compare the performance of the LSC and BSC, we considered the time it took to pebble PTC graphs constructed with both families of SCs. Our implementation of pebbling stores the graph structure in one file, the hashes of each node in another file. In Section 4, we use these pebbling times to evaluate LSCs, which, to the best of our knowledge, we are the first to implement.

3.2 Simulation of Consensus on Network

The other main contribution of our work was the simulation of the protocol described in Section 2.5 in a network. To do this, we made use of Mininet, a software that creates a virtual network on a single machine [6]. Mininet provides a Python API to create customized networks, enabling the user to set options such as the network's topology, the bandwidth of each connection, and the amount of CPU to dedicate to each miner.

For our network setup, we followed Bitcoin's peer to peer network in which each miner forms long-lived outgoing connections to eight randomly selected peers [3]. Each connection in the network was also assigned a bandwidth of 100Mb/s and a delay of 50ms, in an attempt to best simulate the time propagation would take in a physical network. We also used Python's XMLRPC library to communicate between miners for proving and verifying.

For comparision purposes, we ran seperate tests for reaching consensus with PoW and with PoS. For our PoS tests, we closely followed the protocol described in Section 2.5. The only difference between the protocol and the tests was that there were no transactions or other block data – instead, the miners submitted only their PoS, and continually broadcasted the best proof they have seen so far. For our PoW simulation, we followed the protocol described in [8]; the only major differences between the two in code were the sending and verifying of proofs – the proof being sent in the PoS simulation was just one hash, and verification essentially consisted of hashing the proof. We also exclude the block data from the PoW tests in order to have a fair comparision between the two.

We ran these tests on networks of varying sizes, starting at 25 miners and going up to 150 miners. The main focus was timing the propagation of a single proof throughout the network; to do so, we recorded times at which certain percentages of the network agreed on a single proof. The percentages were maintained by an external host on the network serving as a pollster, keeping track of how many miners agreed on any one of the proofs in real time.

Prior to each test, the miners were tasked with generating a Proof of Space or a Proof of Work for a PTC(16) graph, which equates to about 1.14GB of data. After all the miners have been initialized

and have created their own proofs, each miner would then broadcast their own proof, replacing it with proofs of higher quality as they encountered them from their neighbors. To account for time taken to verify incoming proofs, we also added a pause corresponding to the average time it took to verify a PoS or a PoW for a PTC(16) graph, both obtained in Section 4. This verification and propagation of proofs continued until 100% of the nodes agreed upon a single proof, at which point the test ended. The entire test simulates the addition of a new block to the blockchain.

To the best of our knowledge, we are the first to simulate consensus with PoS. We use our results to support the viability of Proof of Space for distributed consensus in the next section.

4 Results and Analysis

In this section, we present and analyze data on the runtime of pebbling both linear and butterfly PTC graphs, the time to create a Merkle tree, and the time to reach consensus with Proof of Space.

Figure 6 provides the time to pebble PTC graphs recursively defined with both BSCs and LSCs. Note that the number of vertices in the graph is the number of bytes divided by 64, as each vertex stores a 64-byte hash from SHA-512. Also note that this graph is on a log scale, and we only show graphs larger than one megabyte. Finally, note that we later translated our pebbling code from python to c++ and observed a 25 % increase in speed for PTC graphs recursively defined with BSCs. We believe similar performance increases can be done with the PTC graphs that are recursively defined by LSCs, however here we present the complete test results of our Python implementation. After performing a linear regression on the log graph, we find for the butterfly PTC graph that $\log_2(T) \approx 1.01 \cdot \log_2(S) - 21.072$, while for the linear PTC graph, $\log_2(T) \approx 1.022 \cdot \log_2(S) - 20.589$, where T is the time-to-pebble in seconds and S is the size of the PTC graph in bytes. If we substitute $S = 1000000000$, we find that for a 1 gigabyte graph ($\log_2(S) = 30$), pebbling the linear PTC graph is $2^{0.843} \approx 1.8$ times slower than pebbling the butterfly PTC graph. There are a couple of reasons why pebbling the PTC graph with LSCs is slower than pebbling the PTC graph

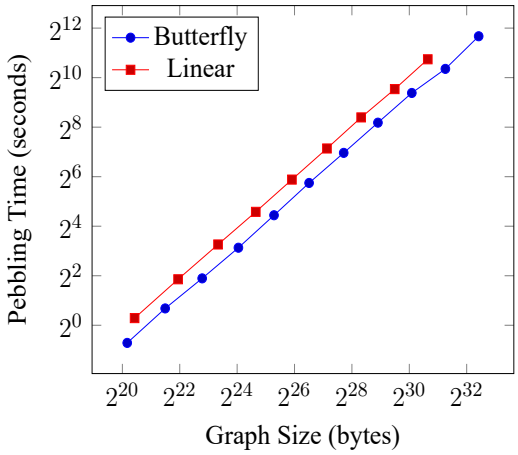

Figure 6: Pebbling Time for PTC Graphs

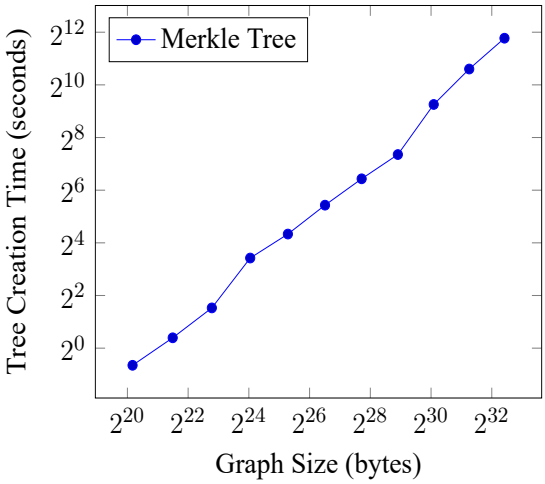

Figure 7: Merkle Tree Creation Time

with BSCs. First, because the vertices have more parents on average (up to seven in the LSC, but only up to two in the BSC), more file reads are necessary to read the hash values of the parents. Second, since there are more parents, the preimage of the hash will be longer, which will slow down the pebbling process more as SHA-512 runs slower for larger inputs.

Recall that the lower bound for the number of pebbles necessary to pebble an n-vertex LSC is $\Theta\left(\frac{n}{\log_2 n}\right)$ [10], whereas the lower bound for the n-vertex BSC is only $\Theta\left(\frac{n}{\log_2^2 n}\right)$ [12]. Moreover, because the constant factors behind the two lower bounds are equal [12], the LSC comes with a lower bound $\log_2 n$ times greater than does the BSC. Because our experiments reveal less than a factor of 2 difference in real-world performance, it follows that the LSC is superior in terms of the difference between the PoS lower bound and the actual pebbling time. This is important because any PoS cheater, no matter how clever, must use at least the space dictated by the lower bound. Consequently, the LSC provides more security at relatively little additional computational cost.

Figure 7 provides the time to create Merkle trees from the pebbled PTC graphs (with BSCs). Note that the x axis is the size of the PTC graph in bytes, rather than the size of the Merkle tree. In general, however, the number of nodes in the Merkle tree is simply $2^{\lceil \log_2(N) \rceil + 1} - 1$, where N is the number of nodes in the PTC graph. Note that we have to round up to the nearest power of two

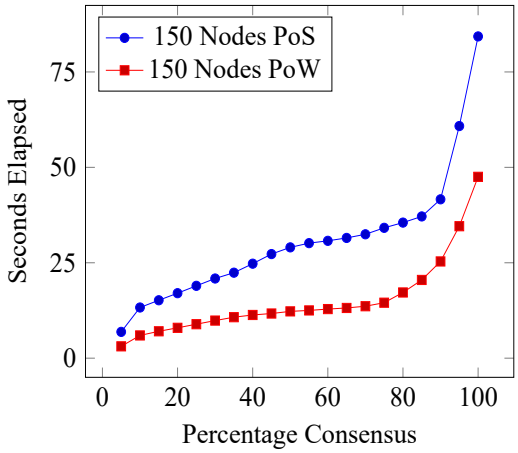

Figure 8: Consensus Time by Percent

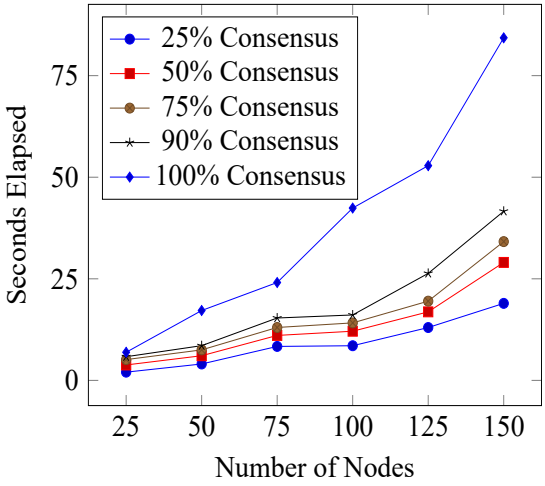

Figure 9: Consensus Time by # of Nodes

in order to implicitly store the binary tree as an array. This causes some irregularities in the graph, making the graph jump every once in a while. We find that the Merkle tree generation time takes roughly as long as the butterfly PTC pebbling time. For PTC(16), which is 1.14 gigabytes, Merkle tree generation takes 47.8% of the time to pebble the PTC graph and construct the Merkle tree.

Figures 8 and 9 provide the time to reach consensus on a peer-to-peer network using a Proof of Space that is 1.14 gigabytes. We use as a conservative estimate for both PoS generation and verification time 0.09 seconds, which is the time it takes experimentally for the entire prover/verifier protocol (the two processes combined).

To capture any experimental biases, we also run the analogous 150-node experiment for the Bitcoin consensus protocol, in which verification requires a single hash and proof consists of only a single value to be hashed, rather than the many hashes used in PoS (Figure 8). We set the PoW verification time to be 0.0005 seconds. Since the experimental consensus time for PoS is within around a factor of two of the time for Proof of Work, and because the times measured are well within the consensus window of ten minutes used for Bitcoin (which is, however, a larger network) [8], we conclude that PoS is a viable alternative to Proof of Work for consensus protocol.

Figure 9 can be used to understand how consensus time for PoS using butterfly superconcentrators scales with the number of nodes. Interestingly, 100% consensus seems to scale considerably

worse than smaller percentages. This is fine, as 90% consensus is sufficient for consensus protocol.

5 Conclusion and Future Work

In our work, we implemented a full prover and verifier interface, in which the prover pebbles PTC graphs, creates a Merkle tree using the pebbled graph as the leaves of the tree, and then opens leaves in the Merkle tree to prove to a verifier that the prover has completed the Proof of Space. We have also implemented PTC graphs with linear superconcentrators, which take roughly 1.8 times longer to generate than those with butterfly superconcentrators, but which provide a better theoretical bound on the minimum space required to pebble the graph. We have also concluded that Proof of Space is viable for distributed consensus on a peer-to-peer network, as our tests with networks indicate that consensus can be reached relatively quickly.

One direction for future work would be to test larger PTC graphs and consensus on larger networks. We could also test consensus with the presence of malicious miners. Finally, we would like to extend all of our work to a full implementation of a blockchain in a cryptocurrency similar to Bitcoin, but using Proof of Space instead of Proof of Work.

6 Acknowledgements

We would like to acknowledge and thank our mentors Albert Kwon and Ling Ren for their invaluable guidance throughout this project.

We would also like to thank MIT PRIMES, Dr. Slava Gerovitch, and Dr. Srini Devadas for providing us with this opportunity to conduct original research.

References

[1] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference*, pages 139–147. Springer, 1992.

[2] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak. Proofs of space. In *Annual Cryptology Conference*, pages 585–605. Springer, 2015.

[3] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 129–144, 2015.

[4] M. Jakobsson and A. Juels. Proofs of work and bread pudding protocols. In *Secure Information Networks*, pages 258–272. Springer, 1999.

[5] N. P. Karvelas and A. Kiayias. Efficient proofs of secure erasure. In *International Conference on Security and Cryptography for Networks*, pages 520–537. Springer, 2014.

[6] B. Lantz, B. Heller, N. Handigol, and V. Jeyakumar. Mininet: An instant virtual network on your laptop (or other pc). http://mininet.org/.

[7] R. C. Merkle. A certified digital signature. In *Conference on the Theory and Application of Cryptology*, pages 218–238. Springer, 1979.

[8] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[9] S. Park, K. Pietrzak, J. Alwen, G. Fuchsbauer, P. Gazi, and A. Kwon. Spacemint: A cryptocurrency based on proofs of space. Technical report, IACR Cryptology ePrint Archive, 2015: 528, 2015.

[10] W. J. Paul, R. E. Tarjan, and J. R. Celoni. Space bounds for a game on graphs. *Mathematical systems theory*, 10(1):239–251, 1976.

[11] S. Radicati. Levenstein. email statistics report. 2015.

[12] L. Ren and S. Devadas. Proof of space from stacked bipartite graphs. Technical report, Cryptology ePrint Archive, Report 2016/333, 2016.

[13] U. Schöning. Smaller superconcentrators of density 28. *Information processing letters*, 98(4):127–129, 2006.