PERFORMANCE ANALYSIS OF PROOF-OF-ELAPSED-TIME (POET) CONSENSUS IN THE SAWTOOTH BLOCKCHAIN FRAMEWORK

by

AMIE CORSO

A THESIS

Presented to the Department of Computer and Information Science
and the Graduate School of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Master of Science

June 2019

THESIS APPROVAL PAGE

Student: Amie Corso

Title: Performance Analysis of Proof-of-Elapsed-Time (PoET) Consensus in the Sawtooth Blockchain Framework

This thesis has been accepted and approved in partial fulfillment of the requirements for the Master of Science degree in the Department of Computer and Information Science by:

Joeseph Sventek	Chairperson
Hank Childs	Member

and

Janet Woodruff-Borden	Vice Provost and Dean of the Graduate School

Original approval signatures are on file with the University of Oregon Graduate School.

Degree awarded June 2019

THESIS ABSTRACT

Amie Corso

Master of Science

Department of Computer and Information Science

June 2019

Title: Performance Analysis of Proof-of-Elapsed-Time (PoET) Consensus in the Sawtooth Blockchain Framework

Blockchains are distributed ledgers that use a tamper-sensitive, append-only data structure in conjunction with a consensus protocol to enable mutually distrusting parties to maintain a global set of states. A primary barrier to adoption of blockchain technology by industry is the current performance and scalability limitations of these systems, which lag far behind incumbent database systems. Of particular interest are "lottery-style" consensus algorithms, which are relatively scalable to many participants but suffer from low throughput (performance). Proof-of-Elapsed-Time (PoET) is one such algorithm with great promise for use in industry, though the parameters that govern its performance have not been well studied. This thesis explores, through simulation, key performance outcomes in PoET blockchain networks implemented with the Hyperledger Sawtooth framework. A better quantitative understanding of the interactions among these system parameters will be crucial for efficiently optimizing real world blockchain networks and facilitating adoption by industry.

CURRICULUM VITAE

NAME OF AUTHOR: Amie Corso

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon, Eugene
Bowdoin College, Brunswick, Maine

DEGREES AWARDED:

Master of Science, Computer and Information Science, 2019,
University of Oregon
Bachelor of Science, Chemistry and Environmental Studies, 2012,
Bowdoin College

AREAS OF SPECIAL INTEREST:

Distributed Consensus
Blockchain Technology
Performance Analysis

PROFESSIONAL EXPERIENCE:

Graduate Teaching Fellow, University of Oregon, 2017 – 2018

Cycling Guide, Trek Travel, 2014 - 2016

Business Analyst, Suterra LLC, 2012 – 2013

Undergraduate Researcher, Bowdoin College Department of Chemistry, 2011

GRANTS, AWARDS, AND HONORS:

Best Graduate Teaching Fellow, University of Oregon, 2018

ACKNOWLEDGMENTS

My sincerest gratitude to our Department Head Joseph Sventek, who recognized that I was ready for a challenge and made this research opportunity available to me, supporting me throughout the year despite the many demands on his time. I would also like to thank Professor Hank Childs for his encouragement in research, participation on my committee, and thorough edits of this document. I thank my brother Anthony for his commiseration through graduate school and his fountain of good ideas. Finally, many thanks to my parents and friends for their endless patience, encouragement, and faith in my abilities.

TABLE OF CONTENTS

Chapter Page

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

Despite the recent, extraordinary hype that has swept both the technological and popular spheres, "blockchain" technology isn't a radically new innovation. Drawing upon advances in distributed computing, cryptography, game theory and other fields, blockchain technology is a novel synthesis of prior concepts [1]. At a high level, blockchain technology can be thought of as a protocol that enables coordination and consensus among a group of participants, connected across the Internet. Participating peers collectively maintain a log (the blockchain itself) and use the consensus protocol to determine what will be appended to this data structure. A blockchain is commonly referred to more generally as a "distributed ledger" because the data that is actually recorded on the blockchain is transactional. Like a traditional ledger, each entry in a blockchain represents some change of state to whatever is being managed by the blockchain network. Currency is a simple example of what might be implemented in a distributed ledger; a particular state is the set of all user accounts and the balance of each account. A transaction recorded on the blockchain specifies a modification to this state by indicating a transfer of currency between two accounts.

Being such a straightforward and relevant application of a distributed ledger, it's no surprise that a currency was the first successfully demonstrated use case of this technology. In October of 2008, the whitepaper outlining Bitcoin's Proof-of-Work (PoW) consensus protocol was published pseudonymously by Satoshi Nakamoto [2]. In the following January, the first version of Bitcoin was deployed and the first bitcoin created, marking the beginning of the first public, decentralized, cryptographic currency [3]. Consensus algorithms have been extensively studied for decades by the distributed systems community, and many robust algorithms exist that can tolerate various degrees of dysfunction among the nodes, including faulty and malicious peers. However, most prior work in distributed consensus has been focused on closed groups. Bitcoin's Proof-of-Work protocol was innovative by making consensus possible in a completely open, anonymous network. This, coupled with the use of a cryptographically secure data structure (the blockchain) is what sets blockchain technology apart from its precursors.

At first, Bitcoin was slow to gain attention. For about four years, the price of Bitcoin lingered between a few cents and roughly $10, before gaining traction in 2013; Bitcoin eventually spiked in 2017 to peak at almost $20,000 on the first day of 2018 [4]. At the time of writing in mid-2019, the price of Bitcoin has stabilized around $5,000 for the duration of 2019. The public awareness of blockchain technology has followed a similar trajectory. In particular, it became clear that this technology was capable of supporting distributed applications that are far more complex than simply pushing cryptocurrency between accounts. In 2013 Vitalik Buterin began sharing a whitepaper outlining the idea for a "Turing-complete, general-purpose blockchain" which, in 2015 and with the help of fellow developer Gavin Wood, became the Ethereum blockchain [5]. Ethereum runs a virtual machine (the Ethereum Virtual Machine or EVM) that interprets transactions and executes "smart contracts" – computer code that is stored on the blockchain itself. This means any deterministic logic can be implemented on the Ethereum blockchain and then invoked using transactions. While Ethereum supports a cryptocurrency called Ether, this currency is intended as a utility currency to pay for the use of the EVM and meter computing resources. Smart contracts are extremely diverse – they can be used to surround financial transactions with arbitrary logic, to govern Decentralized Autonomous Organizations (DAOs), to fundraise for charity or startups, and even to facilitate digital collectibles as in the CryptoKitty craze [6].

As a general-purpose computing platform, Ethereum has become an example of the extremely diverse types of applications that can benefit from the properties of blockchain technology. Blockchain technology is being heralded by many as comparable in revolutionary potential to the Internet, and many new designs are being developed that differ considerably from Bitcoin and Ethereum.

1. The Hyperledger Project

In response to the rapid increase in attention from developers and the burgeoning diversity of blockchain implementations, The Linux Foundation in late 2015 established the Hyperledger Project - "an open source collaborative effort created to advance cross-industry blockchain technologies" [7]. Hyperledger is an umbrella project, with (as of

2019) twelve independent blockchain-related frameworks and tools in its portfolio. With significant experience managing open source projects, the Linux Foundation has partnered with industry leaders such as IBM, Intel and over 250 additional member organizations to make Hyperledger its fastest-growing open source project [8]. Intended for enterprise use cases, the Hyperledger projects are completely dissociated from cryptocurrency. The Hyperledger frameworks are designed to be use-case customizable and deployed in business networks, as opposed to the massive, public, single-instance Bitcoin and Ethereum networks. The six blockchain frameworks in the Hyperledger umbrella are called Fabric, Sawtooth, Indy, Iroha, Grid and Burrow. The other six projects are auxiliary tools: Caliper, Cello, Composer, Explorer, Quilt, and Ursa. These projects offer support for performance analysis, network visualization, ledger interoperability, cryptographic libraries and more.

Taken collectively, the Hyperledger frameworks are beginning to address most of the major proposed use cases for blockchain technology, including the creation of secure digital identities, management of physical assets or intellectual property, improved voting and governance, streamlined clearing and settlement, automation of business logic and contract fulfillment, and the improvement of supply chain management and transparency. It is this final case – improving the management of global supply chains – that was the original inspiration for the second project to enter the Hyperledger incubator, Intel's Sawtooth Lake blockchain project (now known as "Hyperledger Sawtooth" or "Sawtooth"), the framework used in this research.

2. Motivation

a. Supply chain management

The global supply chain is a wildly complex, interconnected network of supply chains with no central authority. "Supply-chain management" is the oversight and coordination of supply chain activity, and its objectives include reducing costs, maintaining product quality, moving goods dependably with efficiency, minimizing risks, promoting sustainability, and maintaining flexibility [9]. Consumers and governments have their own interest in robust management, including transparency of product origin

and traceability of goods to aid in preventing food and pharmaceutical safety problems, promoting sustainable consumer decision-making, and reducing illegal or fraudulent activity. Modern supply chain management faces many problems in meeting these objectives that may be eased or eliminated with the help of blockchain technology. Specifically, blockchain technology provides the properties required for independent, mutually distrusting organizations to collectively maintain integrated records. Integrated recordkeeping with existing database technology would require a centralized data store that is susceptible to single-point failure, corruption or fraud, and would have to be managed and financed by a single entity. Blockchain technology is a way to create an integrated database that is equally governed by all participating parties with a high level of security.

Integrated transactional records could make it possible to trace a product all the way from storefront to its sources of raw materials, thus helping consumers make conscious, competitive choices about which industries to support. Better integration would also help pinpoint the source of foodborne illnesses or other safety hazards, and could aid in optimizing the supply chain itself by identifying and removing inefficiencies. Blockchain technology may also prove prerequisite to widespread incorporation of Internet of Things (IoT) technology: widespread, internet-enabled devices used to monitor shipping conditions, delivery times, and product identities. The use of smart contracts (embedded logic) within a supply chain blockchain could allow for trustworthy automation of business logic that has previously been manual.

b. Barriers to adoption

In many industries, including supply chain management, the benefits of blockchain technology won't manifest until there is widespread industry adoption. Unlike other technologies that may be useful in isolation, blockchain technology is only useful insofar as it promotes the cooperation of multiple participating entities. This fact, coupled with corporate technical inertia, poor understanding and strong skepticism from the general public, pose barriers to adoption. Such skepticism isn't unfounded – the major hurdles to widespread adoption lie in the current limited performance and

scalability of these systems. These properties are deeply related to the consensus protocols at the heart of a blockchain's unique and desirable capabilities. There is a tradeoff between the security and cooperation enabled by distributed consensus, and the vastly higher performance and scalability of traditional, centralized database systems. Until blockchain systems can satisfy the incumbent industrial performance standards, their benefits will remain unexploited.

As the styles and implementations of blockchain technology continue to diversify, so does the landscape of design choices and tradeoffs. Some consensus protocols offer high performance (such as high transaction throughput and low latency) but can't scale in size beyond about 12-16 validating nodes [10]. Other protocols may be scalable to vast numbers of such nodes, but suffer poor throughput. Before businesses will be willing to invest in new technology and protocols, they must be able to make informed, data-driven comparisons to arrive at the best design choices for their specific needs. However, the body of performance research in blockchain technology is small, and much of it is specific to the Bitcoin network. The complexity of these systems makes performance research difficult to conduct, and the results can be difficult to compare meaningfully, given the many parameters and environmental factors that govern the behavior of specific blockchain instances.

3. Lottery-style consensus

The truly disruptive power of blockchain technology is its potential to secure completely open networks. While Bitcoin was first to prove this possible, there are a number of problems with the Bitcoin protocol that will need to be improved in its descendants. In particular, the upper bound on Bitcoin's throughput is 7 transactions per second (compared with Visa's average of 10,000 transactions per second), and its Proof-of-Work protocol is highly energy inefficient. Bitcoin has inspired a suite of "Proof-of-X" consensus protocols that attempt to replace "Work" with less energy-intensive alternatives, and generally improve upon the original PoW algorithm [11]. One such descendant is "Proof-of-Elapsed-Time" or "PoET," first implemented in the Hyperledger Sawtooth project. PoET is very similar to PoW, but replaces the useless computational

expense of PoW with a computationally cheap random wait. All of these PoW variants are considered "lottery-style" consensus algorithms because there is an element of randomness that governs who controls the publishing of blocks and therefore the extension of the chain.

PoET stands out as a useful algorithm to investigate for several reasons. It is one of the most robust available implementations of a Proof-of-X algorithm and is production-ready as part of the Sawtooth project. It is also independent of cryptocurrency and therefore a promising solution to industry use cases in which transactions may not be financial. Finally, it is highly parameterizable, unlike the Bitcoin protocol which offers few mechanisms for update and has remained relatively inert since its inception. Important among these lottery-style consensus parameters is the choice of *block interval*, the amount of time that passes between publishing successive blocks to the chain. As will be further discussed in section 2.8, the choice of block interval establishes the theoretical transaction throughput of a blockchain network. More importantly, the relationship between the block interval and the *size* of the network govern the empirical throughput. The theoretical and empirical throughput differ due to the phenomenon of *stale blocks*, valid blocks that are published to the network, but don't ultimately become part of the blockchain. A stale block occurs when more than one node publishes a valid successor block within such a short period of time that they aren't aware of each other's blocks. This leads to a fork in the blockchain, as multiple potential successors are received, validated, and retained by the nodes in the network. The presence of stale blocks ties up processing resources at each node, and slows the growth of the chain, leading to longer effective block intervals and a reduction in throughput. The combination of a short block interval and a large network is likely to give rise to many stale blocks, significantly reducing throughput. However, if the block interval is too long, the throughput is also reduced by a shrinking theoretical upper bound.

The potential for customization makes PoET optimizable and adaptable for its intended use; but, as pointed out by Gervais et al. [12], "although the security provisions of Bitcoin have been thoroughly analyzed, the security guarantees of variant (forked) PoW blockchains (which were instantiated with different parameters) have not received

much attention in the literature." In addition to such security provisions, the performance and scalability of PoW-derivatives have not been thoroughly studied.

4. Research goals

This thesis explores, through simulation, how the key performance outcomes of transaction throughput and stale block rate are affected by both the size of the network and the choice of block interval in the lottery-style consensus algorithm Proof-of-Elapsed-Time. Previous work has examined the effect of different block intervals on security outcomes in several established cryptocurrency networks [12], but do not focus on the performance implications. Another study has explored PoET's throughput and stale block rate in networks of varying size, but does not vary the choice of block interval [10]. This thesis contributes to the growing body of performance analysis research in blockchain technology and lays groundwork for performance optimization of lottery-style consensus algorithms.

5. Outline

Section 2 provides background terminology and concepts in blockchain technology and lottery-style consensus, expounding on the performance properties of these systems and the interactions among key parameters. Section 3 presents related work and challenges inherent to performance analyses of complex systems. Section 4 provides a detailed description of the experimental environment and methods used to simulate Sawtooth networks with PoET consensus, deliver experimental workloads, and collect performance metrics. Section 5 presents the results of these simulations and a discussion of the observed phenomena. Finally, Section 6 outlines several directions for future work including improved simulation design and the potential for formal modeling.

CHAPTER II

BACKGROUND

1. Key properties

The term "blockchain" refers not to a single entity (such as Bitcoin), piece of software, data structure, or algorithm, but rather to a growing collection of various technologies that share certain key properties. Blockchain technology is a subset of "distributed ledger technology" (DLT), which involves the synchronization of storage, access, and maintenance of digital data across a multi-party network of distinct participants. A decentralized ledger is so-called because data is stored and accessed not in a central repository, but is replicated across multiple "nodes" or "peers." The data stored in a blockchain can also be thought of as supporting an abstract finite state machine, a system that exists in exactly one of a finite number of possible states at any given time. The possible states and the valid transitions among them are customizable to the specific application of the blockchain. Thus, blockchains are not restricted to representing simple transactional information, such as transferring currency, but can be capable of executing and storing arbitrarily complex logic and data. As a toy example of arbitrary state representation, Sawtooth developers created a Tic-Tac-Toe transaction family, in which the state of the system is the position of all X's and O's on a game board, which player's turn it is, and whether the game has ended or not.[1]

Distributed ledger technology already faces a handful of difficult problems inherent to distributed systems, in which the network may experience latency or partition, and nodes may fail or commit errors. In particular, a distributed ledger must achieve consistency across replicas and remain live despite the presence of synchronicity challenges. As a subset of DLT, blockchain technology faces the same problems, and tackles several additional ones. In particular, blockchain technology assumes a more hostile security model in which peers on the network may not trust each other and may exhibit intentionally and intelligently malicious, selfish, or destructive behavior in

[1] https://sawtooth.hyperledger.org/docs/core/releases/1.0/transaction_family_specifications/xo_transaction_family.html

addition to benign crashes or random error. To facilitate trustless transactions, the key properties of a blockchain data structure are *immutability*, *transparency*, and *tamper-sensitivity*. Immutability can also be understood as append-only updates; once data has been committed to the blockchain, it cannot be changed without destroying and re-creating the blockchain from that point forward (an extreme version of an append). Transparency means that all validator nodes on the network have access to the same data, which they use to check the validity of updates and collectively police the network. In the case of blockchain technology (as opposed to other distributed database designs) all peers can access their own complete copy of the ledger, such that the state of the ledger and all modifications to that state can be independently verified by all participants. Tamper-sensitivity is a property that arises from the blockchain data structure itself which is a change-sensitive, hash-based structure that allows *efficient* detection of even the tiniest modification.

2. Terms and concepts

The following describes the important terminology and major components of a blockchain data structure and the network that maintains it. The data contained in a blockchain is a sequence of "transactions." A transaction is the method of changing the state of the abstract state machine represented by the blockchain. As a concrete example, consider the cryptocurrency Bitcoin, in which users have accounts that can send and receive bitcoin (the currency of the same name). The state machine represented by the Bitcoin blockchain can be thought of as the balance of all user accounts. A transaction submitted to the Bitcoin network simply specifies a sender, receiver, and amount of Bitcoin to be transferred, thus changing the state of the system by reducing the sender's balance and increasing the recipient's balance by the given amount. The blockchain, as an append-only, immutable structure, doesn't actually maintain a running balance for each account. Rather, the balance of a particular account is the result of every transaction ever applied to that account, and it is only these transactions that are recorded in the blockchain. A "block" is the atomic unit of extending the chain, and is simply a bundle of transactions with some additional metadata required by the consensus protocol, such as

the hash of the previous block, timestamp information, the cryptographic signature of the publisher, and its height on the chain (the number of predecessor blocks).

3. Data structure

A "blockchain" is so named due to the format of the underlying data structure, which largely differentiates a blockchain from other types of distributed database. A key cryptographic technique relied upon heavily in blockchain technology is *hashing*. A *hash function* takes in any piece of data (a stream of bytes) and produces a *hash value* (often a 256-byte hexadecimal string) of that data. Hash functions have several key properties that make them useful. They are one-way, meaning that knowing a hash value does not allow one to reconstruct the original data. They are typically highly change-sensitive, meaning that changing even a single bit of the input data will lead to an entirely different hash value compared to the original. Finally, they are collision-resistant, meaning that it is so unlikely as to be effectively impossible that two different inputs will yield the same hash value. Given these properties, hash values are an efficient way to "fingerprint" any arbitrarily-sized piece of data with a fixed-size hash value.

A blockchain is a form of Merkle tree, a change-sensitive, hash-based data structure in which hash values are successively aggregated (hashes are hashed) and can be used to quickly fingerprint large amounts of data. Transactions belonging to a block are stored as the leaves of a Merkle tree, and the root hash of this Merkle tree is stored in the block as a way of fingerprinting the entire collection of transactions. Sequential blocks are chained together using hash values as well. When a new block is added to the blockchain, part of the data inside the new block is the hash value of the previous block, which in turn contains the hash value of its predecessor, and so on until the "genesis" block, or first block in the chain. By chaining successive hash values, it becomes possible to efficiently verify that the data contained in one replica of the blockchain is exactly identical to the data in another [3, 12]. Finally, the "nonce" field is a random number that, when hashed in conjunction with the block, yields a hash value that is an acceptable solution to the PoW puzzle. The "work" in PoW involves repeatedly guessing and hashing nonce values

until a winning nonce is found, thus completing a valid block. The PoW algorithm will be revisited in section 2.6.

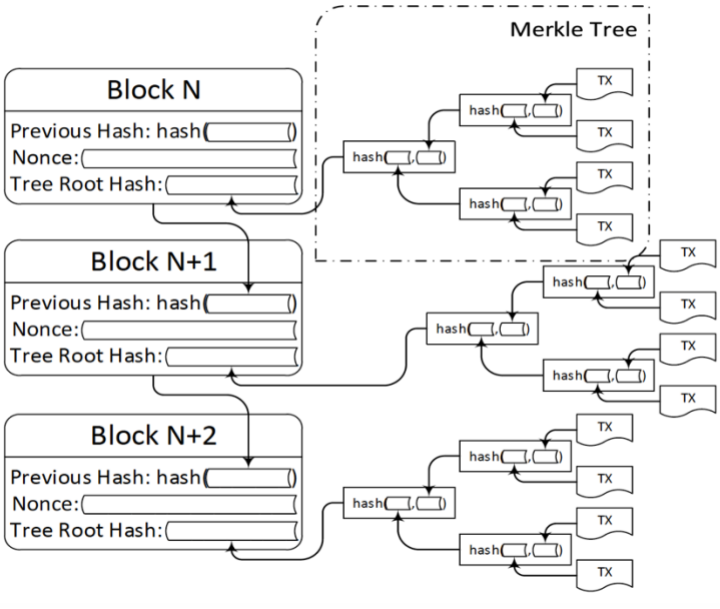

Figure 1: Block structure and Merkle trees [12]

4. Smart contracts

In the Bitcoin example above, a transaction is a simple piece of logic specifying a transfer of value between two parties. However, a blockchain state machine may be capable of representing more complex states than accounts and balances, and capable of moving between such states using complex logic. The implementation of complicated transactional logic has come to be known widely as a "smart contract," coined by the Ethereum network [2], though the same concept goes by other names such as "chaincode" (Hyperledger Fabric [14, 16]) or "transaction processor" (Hyperledger Sawtooth). A smart contract, chaincode, or transaction processor is code that is responsible for interpreting a transaction with the logic intended to govern it, determining whether the transaction is valid, and producing a corresponding change of state. A simple example of what a smart contract might do is control the release of funds until

multiple parties submit approval within some fixed window of time. A more complex usage might involve executing business logic and initiating transactions such as the exchange of physical goods based on the satisfaction of contract conditions.

5. Public vs. private

While any blockchain should provide immutability, transparency and tamper-sensitivity, there are many diverse implementation and design choices that affect the properties and performance of the system. This means that different blockchain systems can be taxonomized along several axes and that the "best" architecture for a given use-case will depend on optimizing for that unique context.

A primary and important distinction is between "public" (or "permissionless") and "private" (or "permissioned") blockchains. Bitcoin is an example of a public blockchain. A public blockchain is so-called because it is truly open to the public – absolutely anyone can submit data to the chain, read data from the chain, and participate in its maintenance. A private blockchain, in comparison, is one that is not open to the public. There are various mechanisms by which access may be restricted, and various types and degrees of privacy, but any chain that isn't publicly accessible is considered private. In the case of private blockchains, the question may arise whether a blockchain is necessary at all. If every participant is already known and verified, isn't the system trusted, as opposed to trustless? In short, not necessarily. This is because multiple parties may be permissioned to contribute to the same blockchain, but may still be business rivals or have other conflicting interests. It is also true that even parties believed to be trustworthy may still act maliciously, and using a blockchain data structure protects against this possibility. Use of a private permissions model may allow for the relaxation of certain security assumptions in the name of performance, while still maintaining security against rare but possible threats. Finally, a permissioned network has the ability to impose size constraints on the network, which may be prerequisite to other design considerations such as the choice of consensus algorithm.

While blockchain technology may have a place in private settings, the ability to secure public networks, as in the case of Bitcoin, has uniquely disruptive potential as a

truly and fully decentralized, anonymous system. As research continues in blockchain network architecture and the technology adapts to a wider variety of use-cases, hybrid private-public architectures may also arise. See [13, 14, 15, 16] for further discussion of permission models in blockchain technology.

6. Consensus

The properties that make blockchain technology powerful, such as data integrity and resilience to malicious nodes, depend primarily on the choice of consensus protocol used to coordinate the network. Distributed consensus has been studied for decades in both centralized and decentralized systems, and there are many robust algorithms for various scenarios. The primary innovation of blockchain technology has been to adapt these algorithms for use in fully decentralized, open networks. Of course, it is still possible to implement a blockchain data structure in a closed network using a traditional consensus algorithm such as Practical Byzantine Fault Tolerance [17], and indeed most of the Hyperledger projects take this approach. These frameworks are intended for use in closed environments, and their primary innovation is the integration of the blockchain data structure with traditional consensus protocols. They are redesigning how data is represented, modified, stored, controlled, and verified, but most of these frameworks remain limited in their ability to secure public networks, or even to support networks larger than 10-16 nodes (compared to the Bitcoin and Ethereum networks which number in the thousands). The following background will therefore focus on the basic concepts of Proof-of-Work consensus and the related Proof-of-Elapsed-Time, the focus of this research.

a. Terms

The nodes that are involved in the consensus process are known as “validators” or “validator nodes” (and in the case of PoW consensus, often as “miners” or “mining nodes”). With regard to security and performance evaluation, the size of a network is only considered as large as the number of validators in the network, since these are the only nodes that participate in consensus and thereby secure the network. In this

document, "node" refers to a validator node unless otherwise specified. A "client" is any node capable of submitting transactions to the network, and may or may not also be a validator. The process of reaching consensus involves broadcasting all data throughout the network and allowing all validators to scrutinize proposed transactions for conformity to network protocol (logic that may be arbitrarily complex, depending on the implementation of the blockchain). Such valid transactions are then prepared to be appended to the blockchain data structure by bundling multiple transactions into a block. The way in which blocks are "published" (submitted for inclusion in the blockchain) depends on the specific consensus algorithm in use, but a published block is once again subject to the scrutiny of the network, and appended to the chain only if it is correct in terms of the relevant protocol. A block is considered "committed" to the blockchain when it has been circulated through the network and applied by all nodes (or by a sufficient threshold of nodes).

b. Security and performance properties

A viable consensus algorithm for a blockchain network has to meet many requirements, the relative importance of which depend largely on the specific use case and deployment environment. First, and most importantly, the algorithm must operate in a fully decentralized manner. There can be no centralized point of failure or corruption; otherwise, the other properties of a blockchain data structure become irrelevant. A single point of centralization renders the entire system centralized — or at least vulnerable in the same way as centralized systems. The algorithm must meet the security requirements of the network under relevant trust assumptions and environment, which in a trustless system means tolerating Byzantine faults – i.e., nodes may exhibit not just crashes or errors, but malicious and strategic behaviors as well [16].

The security solution must not come at the expense of *safety* and *liveness*, terms applicable to any system that deals with concurrency. Safety means, in a general sense, that "bad" outcomes won't happen in a system. In a blockchain network, this means that honest nodes sharing the same protocol and seeing the same data will agree on the same state. In other words, transactions and the state changes they cause are ordered and

deterministic. Liveness means, generally, that *something* will eventually happen (deadlock or starvation will be avoided). In a blockchain network, this means that all nodes must eventually agree on state ("eventual consistency") and that progress will not stall indefinitely. An influential result in distributed systems theory known as the "Fischer-Lynch-Patterson (FLP) impossibility" states that it is impossible to theoretically guarantee both safety and liveness at the same time *in a fully asynchronous system* [18]. In practice, however, it is possible to develop performant and provable guarantees of both *given stronger assumptions*, such as time bounds on message delays and access by nodes to synchronized clocks. Different consensus algorithms place different emphasis on the relative importance of safety versus liveness, another example of how consensus properties should be customized to the particular needs and configurations of a given network.

c. Proof-of-Work: The algorithm

The following describes the basic mechanics of Bitcoin's Proof-of-Work (PoW) algorithm. The Bitcoin network is constantly receiving incoming transactions that are submitted by client applications to nodes throughout the network. Every transaction is broadcast throughout the network, eventually to be received by all validators. Validators assemble these uncommitted transactions into blocks, verifying the validity of each transaction and candidate block, based on the criteria for validity in the Bitcoin protocol. For example, a transaction can only spend bitcoin to which it can prove it has the private key. The validator then hopes to publish their newly assembled candidate block, because there is a financial reward for successfully publishing a block; the publisher gains ownership of the newly minted bitcoin in the block and the transaction fees.

However, in order to publish a candidate block, a validator must solve a cryptographic challenge associated with that specific block. This cryptographic challenge is useless in nature – it adds no information to the blockchain and performs no useful work. The puzzle is simply a game of guess-and-check ("mining"), in which miners systematically search a space of random values for one which, when hashed together with the block, produces a hash value that meets an established requirement (in

Bitcoin, a specific number of leading zeros, also referred to as the "difficulty"). There is no way to gain an advantage in this random puzzle apart from devoting more computational horsepower to try more numbers. If a miner happens upon this satisfactory value before other miners, she will get to extend the chain with her block, reap the rewards, and the race starts again for the next block.

d. Proof-of-Work: Forks and finality

PoW offers only weak consistency – it is possible that different nodes have a different view of the system at the same time. As blocks are published and committed to the chain, it is possible that more than one valid successor block is published by different nodes within a short period of time. In this case, part of the network may first receive and start building upon one block, while the rest of the network may first receive another. This creates a "fork" in the chain, in which two or more equally valid versions of the blockchain exist concurrently in the network. Forks are resolved as one of the branches inevitably grows faster than the other, and the nodes of the network adopt this longest chain as the true blockchain, abandoning the shorter branches.[2]

Therefore, there is a distinction in PoW consensus between a transaction or block being "committed," and being "finalized." A transaction is committed as soon as it is incorporated in a valid block and that block is published. It is only considered final – and therefore valid to reference in subsequent transactions – once buried under enough subsequent commits that the probability of a different fork becoming longer is acceptably small. Of course, a blockchain is an immutable, append-only data structure, so "removal" of a transaction from the blockchain is equivalent to a longer fork of the chain being established in which the transaction is not included.

[2] The choice of "longest chain" is a criterion that would be defined by the specific consensus protocol that the network is using. In some protocols, "longest" may be determined simply by the number of blocks, whereas others might use a slightly different criterion such as "heaviest" with respect to some cumulative resource like work or time.

e. Proof-of-Work: 51% attacks

There are many ways in which a blockchain network can be attacked. This discussion will ignore denial-of-service and other attacks that are purely destructive in nature to focus on the way in which a selfish node might reap value from deception. It is important to note that the Bitcoin software protocol, run by all honest nodes on the network, regards the *longest* valid chain to be the "truth," i.e. the universally accepted version of the blockchain. Therefore, malicious actors seeking to write a history that benefits themselves can only do so in one way: control the longest chain. Only by creating a branch of the chain that becomes and remains the longest can a version of reality be established that is *also* accepted by the network at large. This is the basis for a "double-spend" attack in a cryptocurrency network: a malicious node spends their bitcoin (presumably obtaining something of value in exchange) and the transaction is recorded to the blockchain. The same actor then begins building on a fork of the blockchain that diverges *before* the block that spent the bitcoin, and does not include this spending transaction. If the new fork can grow faster than the original fork, it will become the accepted blockchain, effectively erasing the original expenditure of that bitcoin and leaving it in the hands of its original owner for use elsewhere.

So how does the useless computational work in PoW keep the blockchain secure against this type of transaction reversal? The computationally intensive puzzle associated with each block makes extending the chain difficult and expensive. With a network of several thousand mining nodes around the world racing to publish each additional block, it becomes unlikely that a single node or small group of nodes will be able to publish blocks faster than the rest of the network combined (thus writing the longest chain). In this way, the right to actually record data in the blockchain remains randomized and dispersed across many miners, and the ability of individuals or small subgroups to control the extension of the chain becomes limited by the expensiveness of computational resources. If a single party could control >50% of mining resources in the network, they would, on average, be able to publish blocks faster than any other subgroup of the network and would gain control of what is recorded. This is the nature of a "51% attack," which destroys the distributed stability of the network.

f. Proof-of-Work: Problems

While the Bitcoin network has proven secure in practice, the validation network is not as decentralized, and therefore not as robust to aggregated attacks, as it was intended to be in theory. To gain a computational advantage in the PoW algorithm, miners have created ASIC (application-specific integrated circuit) hardware designed specifically for the PoW computations. General-purpose CPUs and GPUs are no match for the speed of this specialized hardware, and as such, mining Bitcoin has become a capital-intensive activity that encourages the aggregation of mining behavior into "pools." Thus, the Bitcoin validation network is more of an oligarchy than the democracy it was intended to be. Other PoW-based networks such as Ethereum have created variations of PoW to render the algorithm ASIC-resistant.

Another major disadvantage of PoW consensus is the electricity spent performing the computational work. At the time of writing, the estimated annual energy consumption of the Bitcoin network is about 64 TWh or 0.29% of global energy consumption - roughly the same amount of energy used annually by the nation of Switzerland [4, 19]. Even more alarming than this figure is that energy consumption has increased exponentially since Bitcoin's creation and will continue to grow as the network grows. This has to do with the aforementioned need to maintain a rather long block interval. In Bitcoin this interval is approximately 10 minutes, meaning it should take an average of 10 minutes to find the solution to the computational puzzle. The speed with which a solution is found is related to both the difficulty of the puzzle and the computational resources available for solving it. As the network grows, the available computational resources grow, and therefore the difficulty of the puzzle is increased in order to maintain the consistent 10-minute block interval. In short, the amount of electricity needed to secure the network is proportional to the size of the network, and that represents a major problem in sustainable scalability.

Finally, there is a maximum throughput of (on average) 7 transactions per second in the Bitcoin network. The combination of a 10-minute block interval imposed by the Bitcoin protocol with a fixed maximum block size creates this upper bound. As will be

discussed, throughput is the primary challenge in all PoW systems, but Bitcoin's fixed protocol offers little flexibility to improve throughput in that network.

g. Proof-of-Elapsed-Time (PoET): The algorithm

Proof of Elapsed Time (PoET) is a promising, novel consensus algorithm developed recently by Intel in conjunction with their Software Guard Extension (SGX)[3] technology, and implemented in the open source Hyperledger Sawtooth blockchain framework. PoET is a form of PoW consensus, but aims to eliminate the wasteful energy consumption associated with the original algorithm.

Traditional PoW can be thought of as a lottery where the chance of winning is proportional to the amount of computational work expended. The winner is non-deterministic – while the likelihood of winning is proportional to the computational investment, any node may be the first one to find a solution and publish the block. This randomness helps prevent any one party from systematically controlling the writing of blocks. PoET creates a similar lottery-based system for block publishing, but instead of spinning its computational wheels, each node is assigned a random wait time sampled from an exponential distribution, generated by code running inside a "trusted execution environment" (TEE). All validators assemble transactions into candidate blocks, and as soon as its wait time expires (if no other block has already been published at this height) the node publishes its candidate block and broadcasts to the network. In the case that more than one block is published almost simultaneously, a measure of time-spent-waiting is used to resolve the potential fork, favoring the chain with lowest aggregate wait time.

In PoW, there are no shortcuts to solving the hash puzzle, which can be solved by brute force alone, making it impossible to fake the mining process. In PoET, how can a malicious node be prevented from faking short wait times and thereby controlling the chain? Intel SGX technology makes possible hardware-assisted TEEs, secure areas of a main processor that protect application level code even from processes running at a higher privilege level. TEEs are execution environments that have an extremely small

[3] https://software.intel.com/en-us/sgx

attack surface and are equipped with cryptographic verification procedures that can provide externally verifiable attestations and tamper evidence. The cryptographic attestations produced by the TEE verify that specific code was executed within the TEE, and that the result has not been tampered with. Each node that participates in PoET SGX must be equipped with this specialized hardware. The random wait time is generated from a sufficient source of entropy within the TEE, and cryptographically signed by the TEE such that the validity of the wait time is verifiable by other nodes running the PoET protocol. While the details of TEE security are outside the scope of this document, there are theoretical attacks that can be made on a TEE. In the case of compromised TEEs, PoET implements a second line of defense: a statistical z-test that examines whether any given node's rate of publishing is anomalously fast in a statistically significant way. Therefore, even if a node manages to fool the network with regard to its wait time attestations, they will be prevented from publishing too many blocks by statistics visible to the rest of the network.

There is also an implementation of "PoET Simulator" available in the Sawtooth framework, which is identical in functionality to PoET SGX but does not require SGX-enabled hardware. Instead, the TEE and its cryptographic attestations are simulated but meaningless. PoET Simulator is useful for testing and experimentation when SGX hardware is unavailable. It can also be used in a closed production network where node identities are known and controlled, but would be insufficient to secure an open network. See [5, 7, 9, 21] for further discussion of PoET.

7. Hyperledger Sawtooth

Hyperledger Sawtooth originated as Intel's distributed ledger project, designed to take advantage of their SGX technology, and was the second project to enter the Hyperledger Incubator. The intended use-case for Sawtooth is supply chain management, though it can be customized to implement any transactional logic.

Transactions in Sawtooth are defined by the "transaction family" to which they belong, which specifies the corresponding transaction processor responsible for the transaction's execution. A transaction processor is Sawtooth's analogue to the Ethereum

"smart contract," and is the logic by which a transaction is handled and corresponding updates to global state are made. Unlike smart contracts, transaction processors are not actually written to the blockchain; rather, every participating Sawtooth node runs all relevant transaction processors in separate processes. Transactions are dispatched by the Sawtooth node to its corresponding processor for execution and validation. Despite residing off-chain, attempts to modify the internal logic of a transaction processor will lead to a result that differs from that achieved by other nodes in the network, and any blocks proposed by the deviant node will be rejected by the network majority.

There are several transaction families included in Sawtooth's core implementation.[4] The Settings and Validator Registry transaction processors are required to be run by all Sawtooth nodes. The Validator Registry and (optionally) Identity families are used to manage network permissions and roles. The BlockInfo family is used to provide information about the blockchain itself. The IntegerKey, XO, and SmallBank transaction families are simple transaction types available for educational, testing, and benchmarking purposes. All other transactional logic is meant to be custom-designed for the particular use-case of a production network. Unlike smart contracts on Ethereum, these custom processors can be written in any common programming language; as long as the serialization and deserialization schemes are correctly implemented and the program is deterministic, the internal logic is language-independent. A Sawtooth administrator determines what states will be represented, the encoded representation of those states, and the valid rules of transition from one state to another.

The massively public Bitcoin and Ethereum networks are legacy-dependent and cumbersome to change. Sawtooth is designed to be adaptable and customizable, better serving diverse needs and use-cases on a variety of scales. With a built-in Settings transaction processor, Sawtooth allows on-chain governance of network-wide settings and policies. Sawtooth is capable of acting as a permissioned network using these on-chain policies, or as a public network with no permission restrictions. The consensus algorithm itself is "pluggable" and can be changed on-the-fly in a live network through a

[4] https://sawtooth.hyperledger.org/docs/core/releases/1.0/transaction_family_specifications.html#

voting protocol. Sawtooth currently supports consensus algorithm implementations of both Simulated and SGX PoET, RAFT (only crash fault tolerant), Devmode (simplified random leader) and Practical Byzantine Fault Tolerance. For example, a network may begin as a small, permissioned network running PBFT, and then transition to PoET as it grows in size.

8. Properties of lottery-style consensus

The two major properties that determine the fitness of a consensus algorithm for its intended use are often termed *performance* and *scalability*. Performance refers to the rate at which data can be added to the blockchain, and is usually concerned with metrics like maximum transaction *throughput* (usually expressed as transactions per second or "tps") and average transaction *latency*. Scalability refers to the ability of the network to perform well as the number of nodes increases. Lottery-style consensus algorithms such as PoW and PoET are considered to have good scalability but relatively poor performance. This contrasts with most classical consensus algorithms that are capable of better performance, but suffer from high-complexity message passing and break down at sizes far smaller than the thousands of nodes that comprise the Bitcoin and Ethereum networks. However, the performance and scalability properties of lottery-style consensus are related to each other, and given the particular requirements of a blockchain instance, can be optimized with regard to one or the other.

a. Performance

The throughput performance is bounded by the combination of *block interval* and *block size*. The block interval is the amount of time that passes between the publishing of subsequent blocks. In lottery-style blockchains, the time at which the next block is published is non-deterministic, and therefore the actual block interval varies from one block to the next. In this case the block interval setting is actually a target interval, and the average of real intervals will converge to this value. The block size is the maximum number of bytes per block, which determines the maximum number of transactions that

can be included in one block. Taken together, these parameters establish an upper bound on transaction throughput:

$$\max \text{throughput} = \frac{txns}{block} \cdot \frac{blocks}{second} = \frac{txns}{second}$$

In the Bitcoin and Ethereum networks, the block intervals are fixed at 10 minutes and 30 seconds, respectively. The oft-cited 7 tps throughput bound in the Bitcoin network is based on this interval and the maximum size of a Bitcoin block.

In addition to bounding the throughput, the choice of block interval has security and performance implications. Recall that a transaction is not considered final until it is buried sufficiently deep in the chain that there is an acceptably small probability of it being undone by a longer fork. As the block interval shrinks, blocks are published more frequently, and the likelihood of forks increases. Therefore, a shorter block interval should mean the required number of confirmations (the depth of a transaction in the chain) must increase to achieve the same degree of resilience against double-spend or other transaction-revocation attacks. In other words, a substantial block interval is precisely what secures a lottery-style network, by making fork-based attacks slow and difficult to achieve.

b. Scalability

The high scalability of lottery-style consensus comes from the fact that by distributing information such as a published block, the network only requires a single round of multicast (each node forwards information to its peers who have not yet received it), which eventually allows the information to be distributed to every node in a fully connected network. Therefore, the time complexity of lottery-style consensus scales sub-linearly with the number of nodes in the network [10]. This contrasts with other classical consensus algorithms such as PBFT which incur messaging complexity that is quadratic in the number of nodes, and have been shown to perform poorly in networks larger than 16 nodes [20].

It is worth noting that there is a necessary relationship between the size of the network and the mechanism used to maintain the desired block interval. As the network

grows, there are more nodes attempting to publish blocks. In a PoW network, this means there is more available computing power for solving the hash puzzle. In PoET, this means there are more wait timers and a higher likelihood of one node receiving a short one. The maintenance of a consistent block interval is achieved within the consensus protocol by analysis of the recent history of block intervals, and using this to infer the size of the network. Automatic adjustments are then made to the relevant consensus parameters in the next round. In PoW, this means adjusting the required difficulty level of the next hash puzzle. In PoET, this means adjusting the parameter of the exponential distribution from which wait times are sampled. This allows the block interval to remain relatively constant despite fluctuations in network size.

c. Stale blocks

Discussion of the block interval's impact on performance requires the definition of another term. A *stale block* is a valid, published block that does not ultimately become part of the longest chain. The *stale block rate* is the number of stale blocks as a percentage of total published blocks. A stale block occurs whenever more than one node publishes a valid block within a sufficiently short period. If a valid successor block has already been published, but has not yet reached part of the network, a node that has not received this block may publish and broadcast its own successor block, leading to a fork in the network. As stated in Gervais et al., "Stale blocks are detrimental to the blockchain's security and performance because they trigger chain forks – an inconsistent state which slows down the growth of the main chain and results in significant performance and security implications" [12]. The effect on performance is due to the overhead of propagating, handling, storing and validating the stale block at each node, which cuts into resources that would otherwise be dedicated to advancing the main chain. If this overhead becomes significant enough, the stale block rate can become a bottleneck to throughput.

The block interval directly affects the stale block rate of the network. A short interval makes it more likely that two or more nodes will publish blocks within a window of time shorter than the propagation latency between them, leading to a stale block.

Because the stale block rate depends on block propagation latency, it is also affected by the block size. The larger the block, the longer it takes not only to propagate on the network layer, but also to validate at each node before forwarding to peers. The longer the propagation latency, the larger the window in which a stale block may be generated.

Finally, it is important to note that the time at which a block of transactions is committed isn't the only event that matters in the lifecycle of a transaction. Recall that finality in lottery-style consensus is probabilistic, not deterministic. A transaction must be buried under subsequent blocks before it is considered safe to be referenced by future transactions, which creates additional latency. Whether this latency represents a problem depends on the use-case of the blockchain. In the case of Bitcoin, 6 block confirmations are required before a transaction can become input to subsequent transactions, creating an effective latency of 60 minutes. Day-to-day financial exchanges are usually completed in a matter of seconds, and there is extensive external infrastructure around the Bitcoin network to make faster transaction confirmation possible for small day-to-day amounts. Shorter block intervals may increase throughput, but render the required number of block confirmations higher to achieve the same degree of security, thus increasing the effective latency. The relationship between block interval, required confirmations, and the security of the blockchain has not been well studied.

CHAPTER III

RELATED WORK

1. Performance analysis

The ability to benchmark performance will be necessary to facilitate adoption by industry and allow businesses to choose optimal solutions for their needs. Given the many design choices and environmental factors that comprise a specific blockchain instance, performance analyses of these systems are difficult to design, implement, and understand. Standard benchmarking techniques need to be established to allow meaningful comparisons among different platforms and configurations, but the complexity of blockchain systems, diversity of designs, and nascent status of the industry makes establishing meaningful comparative benchmarks even more challenging. A paper from the Hyperledger Performance and Scalability Working Group (PSWG) in October 2018 outlines many of these challenges and states that more work in performance evaluation will be a necessary first-step toward establishing meaningful comparative benchmarks [21].

Standardization of terminology is prerequisite to intelligent discussion of performance, and the aforementioned paper by Hyperledger is currently the most complete published attempt. For example, even the term "transaction" becomes complicated in a performance context. In some networks (e.g., Bitcoin), a transaction is the atomic unit of state change, but in other networks (e.g., Sawtooth) transactions are bundled into "batches" in which either all batched transactions are committed, or none are. To further complicate matters, not all transactions are created equal: some may involve complicated smart contract logic that itself becomes a bottleneck to throughput. Therefore, it doesn't suffice to consider only the number of transactions, but potentially also the type when considering a throughput metric such as tps. The term "node" is another example of complicated terminology. A node is an abstract concept: it is a single locus of computation, but doesn't necessarily represent an independent piece of hardware or independent user, and nodes may vary drastically in the computing resources available

to them. Many platforms differentiate nodes into distinct roles, such as load-generating clients, validators, or "ordering service" nodes as in Hyperledger Fabric.

These examples serve to point out that an evaluation metric is meaningless without a complete description of the system being tested. According to the Hyperledger PSWG, such a complete description must include the consensus protocol being used, the geographic distribution of the network (and any other factors relevant to network latency or bottlenecks), the hardware and software environments of all peers, the number and types of nodes involved in various roles, tools used for testing and observation, the type of data store used to represent the blockchain, and the nature of delivered workloads [21].

A 2017 paper, "BLOCKBENCH: A Framework for Analyzing Private Blockchains" [22] provides an excellent discussion of the complications that arise while trying to understand component functionality in a complex system. They identify four primary layers in a blockchain system, each of which has the potential to create performance bottlenecks.

1. The first layer is the consensus protocol, which is responsible for achieving consistency among all nodes. Performance constraints in the consensus layer may be intentional in the algorithm itself (PoW is designed to be slow and computation-heavy for security purposes), or an unfortunate side-effect of expensive, high-complexity protocols such as PBFT with its need for high-volume message passing.
2. The second layer is the data model, or the actual data structure used to store the blockchain. Performance may be hindered by the data model layer if transactions are IO-intensive, requiring many queries of the underlying data structure.
3. The third layer is the execution environment, the environment that executes the logic of smart contracts or chaincode. This execution must be fast, since all validator nodes must execute the logic of every transaction. Ethereum executes smart contract logic using its own blockchain-specific virtual machine (the Ethereum Virtual Machine or EVM), whereas Hyperledger executes chaincode written in one of several common high-level languages inside a Docker image.

4. The fourth and final layer is the application layer, which represents any application that would use a blockchain as its underlying data structure. In the application layer, it's the workflow of the application itself that may be a bottleneck, which is of lesser concern for system benchmarking. Without targeting the performance of a specific layer, it may be unclear where performance bottlenecks are occurring. The BLOCKBENCH framework delivers layer-targeted workloads in order to apply isolated stress to potential bottlenecks.

Performance analysis and benchmarking for blockchain technology is still in its infancy. This is especially true for the many emerging approaches to consensus that differ from traditional PoW, which has thus far received the most attention from researchers. A deep understanding of these systems will come from a combination of empirical evidence, simulation, and theoretical analyses. Expert Christian Cachin of IBM [16] points out that "simulation alone isn't enough to prove the security of a model since truly exhaustive tests are usually impossible in a given scenario space, so we resort to mathematical tools and analysis". At the same time, the extreme number of variables introduced in real-world instantiations may render mathematical analyses insufficient for predicting the actual behavior of a system. It will be important to identify which design factors dominate the performance of the system, in what way, and in which combinations.

2. "On the Security and Performance of Proof of Work Blockchains"

A 2016 paper by Gervais et al. [12] provides a "novel quantitative framework" of what they term "PoW based blockchains," referring to lottery-style consensus variants of Bitcoin's PoW. Specifically, they design a Markov Decision Process to model optimal adversarial strategies in PoW blockchains given a set of parameters including the stale block rate, adversarial mining power and mining costs, required block confirmations, and network connectivity. They incorporate instance-specific parameters such as network delays, block intervals, block sizes, and information propagation mechanisms to estimate (or empirically evaluate) the corresponding stale block rate. "The main output of the blockchain instance is the (measured or simulated) stale (orphan) block rate, which is fed

as input into our security model." They find that 1) the block interval and the block size are the primary factors that affect the stale block rate and 2) the stale block rate can be used as a single metric that largely determines the resilience of a system to double-spend attacks.

This work focuses on three real networks: Bitcoin, Ethereum, and Litecoin, all of which support cryptocurrencies. The construction of their model is based on financial incentives that will not directly translate to a network such as PoET that does not rely on a cryptocurrency. However, the performance implications of their results are relevant to a PoET network which is mechanistically similar to the PoW networks. They evaluate the throughput achieved by several simulated blockchain instantiations in which they vary the block interval from 0.5 seconds to 25 minutes, and the block size from 0.1 MB to 8 MB. They find that increasing the block interval or decreasing the block size both contribute to reducing the stale block rate. There are a number of configurations of these parameters that yield the same degree of security, but have vastly different throughputs, and conclude by stating, "Our results show that there is considerable room to enhance the scalability [meaning throughput, in this context] of existing PoW without significantly compromising security." This paper is primarily focused on the security of cryptocurrency blockchains against double-spend attacks. While their results indicate that short block intervals and large block sizes both lead to a higher number of stale blocks, there is no systematic characterization of their relationship to throughput, or of the relationship between stale block rate and throughput.

3. "Chain of Trust: Can Trusted Hardware Help Scaling Blockchains?"

The only PoET-specific performance analysis available in the literature was performed by researchers at National University of Singapore in 2018 and appears in the paper "Chain of Trust: Can Trusted Hardware Help Scaling Blockchains?" [10]. They analyze the effect of network size on transaction throughput and propose a small modification to PoET (which they term "S-PoET") that makes it possible to maintain higher throughput as the network grows. They note that a stale block will occur

whenever the shortest wait time and another wait time differ by less than the propagation delay of the network. If the network propagation delay is δ, average block time is T and network size is *n*, the expected number of stale blocks $\cong \frac{n\delta}{T}$. This expression obviously increases as a factor of *n*, but perhaps less obviously, δ also increases with network size as it takes longer for a block to reach all nodes that may be geographically dispersed, weakly connected, or busy validating previous stale blocks. This leads to a super-linear increase in the stale block rate as the size of the network increases.

Their experiments were performed using Sawtooth v0.8. 32 physical servers were used to simulate up to 128 Sawtooth nodes running in Ubuntu virtual machines. Though not specified in the paper, the researchers specified via correspondence that each node was provided 64GB of RAM and ran several IntKey transaction processors in parallel (though precisely how many remains unclear). IntKey transactions were chosen as the workload transaction family, which incur minimal processing overhead and can be performed in parallel as they are free of dependencies. Each node was connected to $\sqrt{n}$ other nodes at random. They chose to use 4MB and 8MB blocks and show results for both sizes. The block interval was "varied from 12 to 24 seconds" but it is unclear which choice for this parameter is presented in the experimental results. Shorter block intervals than 12 seconds were not used because the stale block rate quickly became extreme.

The results of these simulations, included here in Figure 2, demonstrate that network throughput decreases significantly as the network size increases, other factors held constant. The stale block rate increases proportionally, and is proposed as the causal explanation for reduced throughput. In an ideal system, a network of any size would process the same number of transactions per second as a single node (assuming the same number of transactions were being delivered). This is because every node in a blockchain network eventually sees and processes the same transactions as every other node. The performance decline witnessed in these results comes from the overhead of handling the stale blocks produced when multiple nodes compete to publish the next successor; the delay incurred while receiving and resolving forks in the network is accumulated, since a fork must be resolved before constructing and awaiting the next

valid successor. A large prevalence of forks in the network causes delays at all nodes, which in turn lengthens the overall intended block interval and restricts throughput.

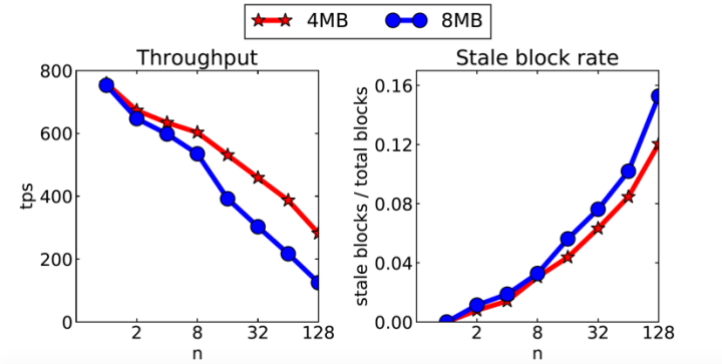

Figure 2: Throughput and stale block rate as a function of network size (n) from Dang et al. [10]

CHAPTER IV

MATERIALS AND METHODS

1. Hyperledger Caliper

Hyperledger Caliper is a performance analysis tool for delivering workloads to backend blockchain instances and collecting a variety of performance metrics. Caliper provides an abstracted interface for the workload specification and performance monitoring that is independent of the specific blockchain instance. Caliper currently supports interfaces for Hyperledger Fabric, Sawtooth, Burrow, Composer and Iroha. Users are able to write custom implementations of any transaction type they may want to use as a Caliper workload.

The workload specification must include the transaction type, either a time duration or a fixed number of transactions to deliver, the "rate controller" parameters (which determine the rate at which transactions are delivered and how they are spaced), and any other information which may be required by the specific backend instance. The blockchain network specification is completely controlled by the user, and Caliper needs only the endpoint URLs for workload delivery and performance monitoring in order to interface with the backend system. When a workload round has been completed, Caliper generates an HTML report of both performance and resource consumption metrics. The specific performance metrics include the absolute number of transactions delivered (both successful and failed), the average, minimum and maximum transaction latency, average transaction throughput, and the actual send rate. The specific resource metrics include average and maximum CPU usage, memory usage, and I/O traffic.

2. Environment

The experimental Sawtooth networks were hosted on a Linux Skull Canyon with 8 cores (Intel Core i7-6770HQ CPUS @ 2.60 GHz, 4 cores with 2 threads per core) and 16 GB of RAM, running Ubuntu 16.04. Given the constrained computing resources on the Skull Canyon, along with the potential for fluctuating resource consumption by the

Caliper workloads themselves, the workloads were delivered to the experimental networks from an instance of Caliper running remotely on an iMac.

3. Network

The highest version of Sawtooth currently supported by Caliper is v1.0.5, whereas the highest production-worthy version is v1.1. In addition to the primary concern of compatibility with Caliper, v1.1 is still immature and subject to bugs and lagging documentation. Therefore, Sawtooth v1.0.5 was used as the backend blockchain instance in these experiments. Sawtooth is designed to be used with Docker containers or to run natively on Ubuntu 16.04. A single Sawtooth "node" is actually a collection of several processes running together: a validator, a REST-API, a validator registry, the settings transaction processor, any other relevant transaction processors for the network, and an optional shell for interacting with Sawtooth via its built-in CLI. Docker containers are a lightweight way to instantiate multiple Sawtooth nodes (and the collection of required processes) on shared hardware. The use of Docker Compose simplifies the process of spinning up and tearing down the collection of node processes, each of which runs in a separate container. It also simplifies the creation of multiple networks that are identical apart from the number of participating nodes. Therefore, the backend Sawtooth instances were hosted in Docker containers instead of run natively on Ubuntu.

Upon configuring a network for deployment, there are many parameters that can be specified to the genesis validator (the validator tasked with creating the first block) upon startup, which establish the initial settings of the chain. Most importantly, the type of consensus algorithm is established upon startup. PoET Simulator was used in these experiments (since PoET SGX requires specific hardware). The only difference in behavior between the PoET Simulator and PoET SGX is the latter requires cryptographic attestation of wait times from the trusted execution environment (hardware enclave); from a performance standpoint, they are the same. PoET is parameterized with a target_wait_time, which establishes the average block interval, and an initial_wait_time, which is used as the target wait time before the network has published a sufficient number of blocks to accurately estimate the size of the network. Designed to allow nodes

to come and go, PoET uses the recent history of block intervals to estimate the size of the network, and then parameterizes the wait time distribution so as to maintain a relatively constant block interval despite fluctuations in network size. When a network is initially established, the initial_wait_time should be explicitly set to target_wait_time*network_size (if the network size is known). With the exception of experiments that intentionally vary the target_wait_time, a value of 20 seconds was used as the block interval for two reasons: the Sawtooth developers suggested using a value at least this high to avoid excessive forking, and the researchers in [10] used block intervals between 12 and 24 seconds (though do not state the precise value represented by their results). The parameter batches_per_block was set to 100, and 20 transactions were bundled per batch, allowing for a maximum of 2,000 transactions per block.

Also by suggestion of the Sawtooth developers, there is a risk in small networks (<10 nodes) that each node may have the opportunity to publish blocks frequently enough that PoET's second-layer z-test defense, with its default settings, may begin rejecting valid blocks. Therefore, in order to disable the z-test, the parameters block_claim_delay, key_block_claim_limit, and ztest_minimum_win_count were set to 1, 100000, and 999999999, respectively. The remaining settings and network configuration details can be found in the Docker Compose file in Appendix B.

4. Workload

As mentioned in section 2.7, there are several transaction families that have been designed and implemented in the core Sawtooth project for testing and educational purposes. The IntegerKey (IntKey) transaction family[5] implements simple key:value storage, in which arbitrary keys can be set to integer values. The IntKey family only supports three operations: set, inc, and dec, which either sets a key to an initial value, or increments/decrements it by a given amount. The IntKey transaction type was chosen for all experimental workloads for several reasons: The work in [10], serving as a starting point for this study, also uses the IntKey transaction family, making an experimental

[5] https://sawtooth.hyperledger.org/docs/core/releases/1.0/transaction_family_specifications/integerkey_transaction_family.html

comparison more useful. Additionally, there are no transactional dependencies between IntKey transactions, which erases any concern that system performance may be impacted by phenomena other than the consensus mechanism itself. This transaction independence also lends itself to potential future work with Sawtooth's ability to use multiple transaction processors in parallel.

Workloads were configured to deliver transactions at a fixed rate and performance data was collected for a spectrum of transaction delivery rates ranging from 5 to 60 transactions per second. Each experiment (consisting of a fixed set of parameters such as network size, delivery rate, block interval) ran for 800 seconds, a timing compromise long enough for approximately 40 blocks worth of data to be included in each throughput estimate, while short enough to collect a high volume of experimental runs. For complete Caliper workload details, see the benchmark configuration file in Appendix A.

5. Data collection

Throughput data was collected from the auto-generated HTML report created after each Caliper run. Caliper's throughput reports were independently verified by querying the blockchain itself with Sawtooth's command-line interface, to obtain the list of blocks and transactions that had been committed during the workload period.[6]

The stale block rate was calculated independently of Caliper and cross-validated in several ways. Using the Docker Python library, the Docker container logs were saved for each validator, for each experimental run. After completing the delivery of a workload, the network was directly queried using the Sawtooth CLI to obtain the official list of blocks on the main chain. All validator logs were then combed to obtain the set of unique block IDs that passed validation in at least one validator. The stale block rate is then calculated as the difference between the total number of valid blocks processed by the network and the number of blocks on the main chain, divided by the total number of valid blocks.

[6] https://sawtooth.hyperledger.org/docs/core/releases/1.0/cli.html

This approach looks at all unique blocks in the system, disregarding whether they were processed by one, several, or all validators. In a second analysis, the stale block rate was calculated per-validator, to observe whether the effective stale block rate as witnessed by an individual validator was the same as the network-wide stale block rate. The SBRs among validators in the same network agree to within at most 2 percentage points of each other. Taking the average of validator's SBRs was statistically indistinguishable from computing the network-wide SBR.

6. Validation

The behavior of the Sawtooth network simulations and Caliper reports were validated with an external script that uses the Sawtooth CLI to directly query the blockchain and observe the committed blocks and their transactions. Additionally, by performing 20 experimental rounds for select configurations of parameters, it was found that the variance in the throughput and stale block rate measurements stabilized with approximately 7 data points. Therefore, 7 experimental runs are averaged in the results below.

CHAPTER V

RESULTS AND DISCUSSION

1. Fixed block interval

Figure 3 displays the average maximum achievable throughput as a function of network size with a block interval of 20 seconds. The stale block rates in the same figure are the stale rates corresponding to the point of maximum throughput. These results follow the same trend as demonstrated in Figure 2, which represent the results from [21] – the highest throughput occurs in a network of size 1, and declines as the network grows

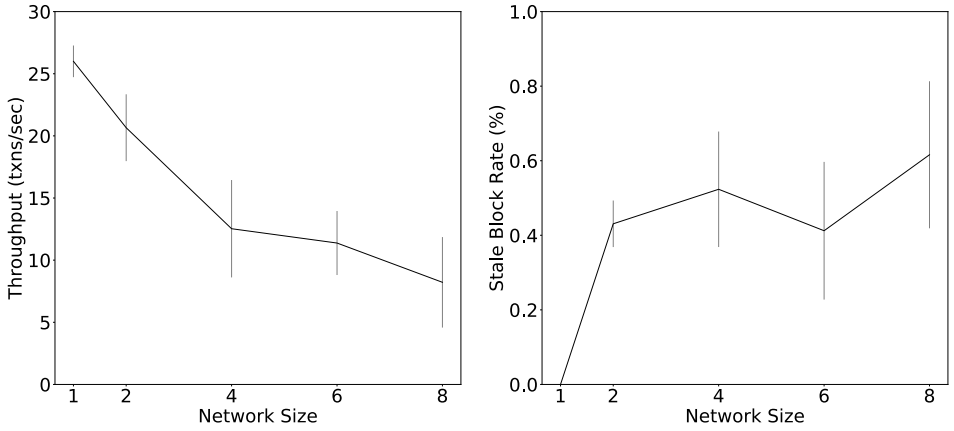

Figure 3: Maximum average throughput and corresponding stale block rate as a function of network size with a 20 second block interval.

grows in size. It's important to note that the computing environments in the respective experiments were significantly different, with the environment in [21] vastly richer in computing resources. Therefore, the absolute numbers are incomparable, but the same relationship holds.

Figure 4 displays several performance metrics: the throughput, stale block rate, number of published blocks, and average block interval, all as a function of transaction delivery rate. At each network size, the throughput peaks close to the equivalent transaction delivery rate, followed by a decline in throughput as the delivery rate exceeds the rate at which the network can process the incoming transactions.

Ideally, the throughput would plateau at the network's maximum capacity, impervious to excess transactions. Instead, there is a notable decline in throughput of about 5 tps, followed by the expected plateau. This performance decline is likely due to

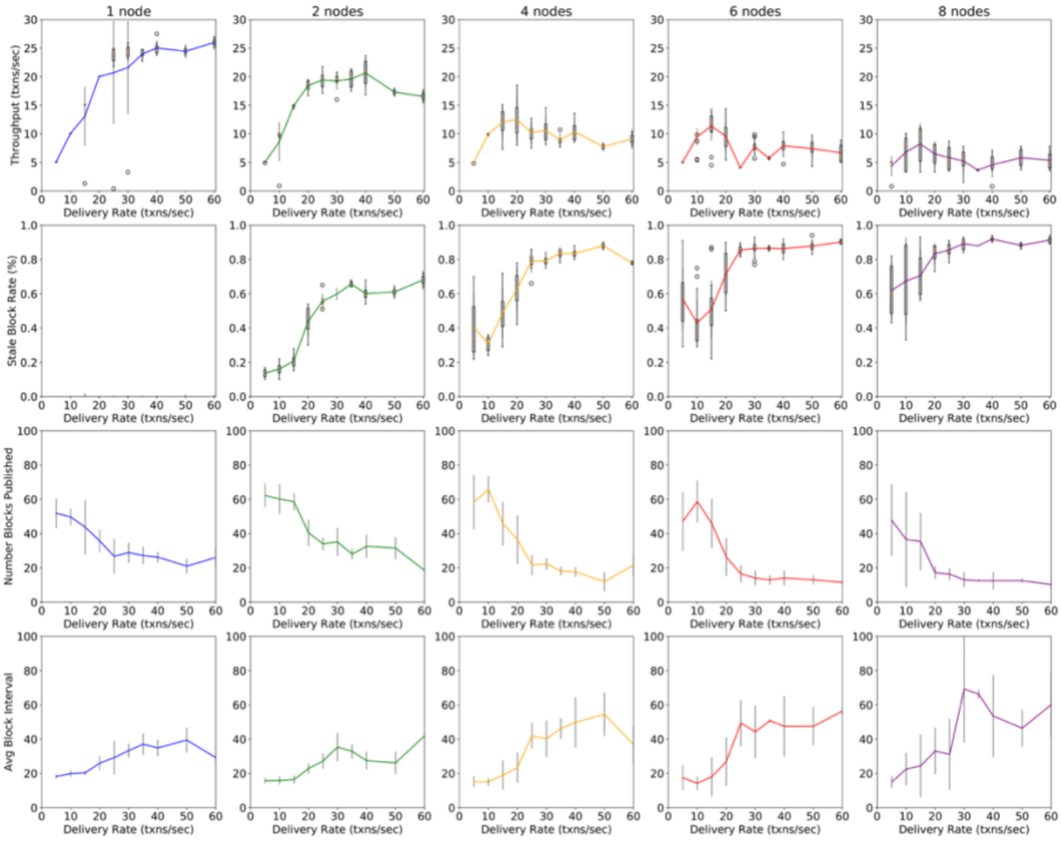

Figure 4: Average throughput, stale block rate, number of published blocks, and real block interval as a function of transaction delivery rate for network sizes 1, 2, 4, 6, and 8.

the communication overhead invoked by a full transaction queue; when a Sawtooth validator's transaction queue fills, each incoming, dropped transaction is acknowledged with a QUEUE_FULL response from the validator, consuming resources that would otherwise be spent in the validation process.

The stale block rates represent the average rate of stale blocks seen across all participating validators. As expected for a single-node network, there is no possibility of stale blocks, and the stale block rate is 0% for all transaction delivery rates. As the network size increases, the stale block rate quickly becomes significant, spiking sharply at the same transaction delivery range in which throughput begins to decline.

There are several notable phenomena to discuss regarding the stale block rate. Most importantly, the fluctuation of stale block with transaction delivery rate demands an explanation. In PoET, blocks are published at roughly the rate of the target interval regardless of whether they are full; as long as there are any transactions to incorporate at all, the validators will proceed with publishing blocks. Therefore, it would be expected that the stale block rate varies with the size of the network (increasing as the network size increases), but not necessarily with the transactional load. At network sizes above 4 nodes, the stale block rate is actually higher at the slow delivery rate of 5 tps, declines around 10-15 tps, and then spikes to a plateau. Both of these phenomena are explained by constrained computational resources in the simulation. The resource monitoring data provided by Caliper indicates that the validator containers reach 100% CPU capacity at the point of maximum throughput, and remain in this stressed state at all higher delivery rates. The extremely high stale block rates may be the result of validators that don't have the resources to simultaneously construct their own blocks while also receiving network traffic and resolving forks. If block creation is prioritized over fork resolution, this will lead to each validator constructing blocks faster than it can integrate incoming information from the network (or than it can broadcast its own). A scenario like this would lead to as many conflicting blocks at each height step of the chain as there are validators.

In the 6- and 8-node networks, the stale block rates plateau at roughly 90%, meaning only 1 in 10 blocks published by the network is actually incorporated into what eventually becomes the longest chain. Using the 8-node network as an example, even if every node published its own block at each successive height, it would be expected that one such block is eventually cemented in the longest chain at that height, while the other 7 became stale, yielding a stale block rate of 87.5%. By the same logic, a 6-node network might be expected to produce stale blocks at a rate of 83.3%, significantly lower than the 92% observed maximum value at this size. Scrutiny of the validators' logs indicate that the forking behavior of the network is not limited to a single height step, but rather forks can consist of multiple blocks at a time. When a fork comparison is made and a new chain head selected, the comparison is based not on the number of blocks (in which two forks may be identical) but on the lowest aggregate wait time of the chain at

the forking height. This means that while a node may be working on extending a particular fork at height *h*, a competing fork may take precedence at a lower height. While the node is then working on extending this new fork, another fork with still higher precedence may arrive and take its place. Thus, it is possible that each node publishes more than one candidate block at a given height in the process of resolving forks.

2. Varied block interval

Figure 5 displays the performance results of varying the target_wait_time from 3 seconds (most green) to 100 seconds (most red) as transactions are sent across the same range of delivery rates. In the single-node network, in which stale blocks cannot occur, a shorter block interval simply means that blocks can be published more frequently, leading to a higher overall throughput. As the network size increases and stale blocks become a factor, this relationship begins to become inverted. In the 6- and 8-node networks, as soon as the transaction delivery rate begins to stress the capacity of the system, the longer block intervals give rise to higher throughput (and correspondingly lower stale block rates).

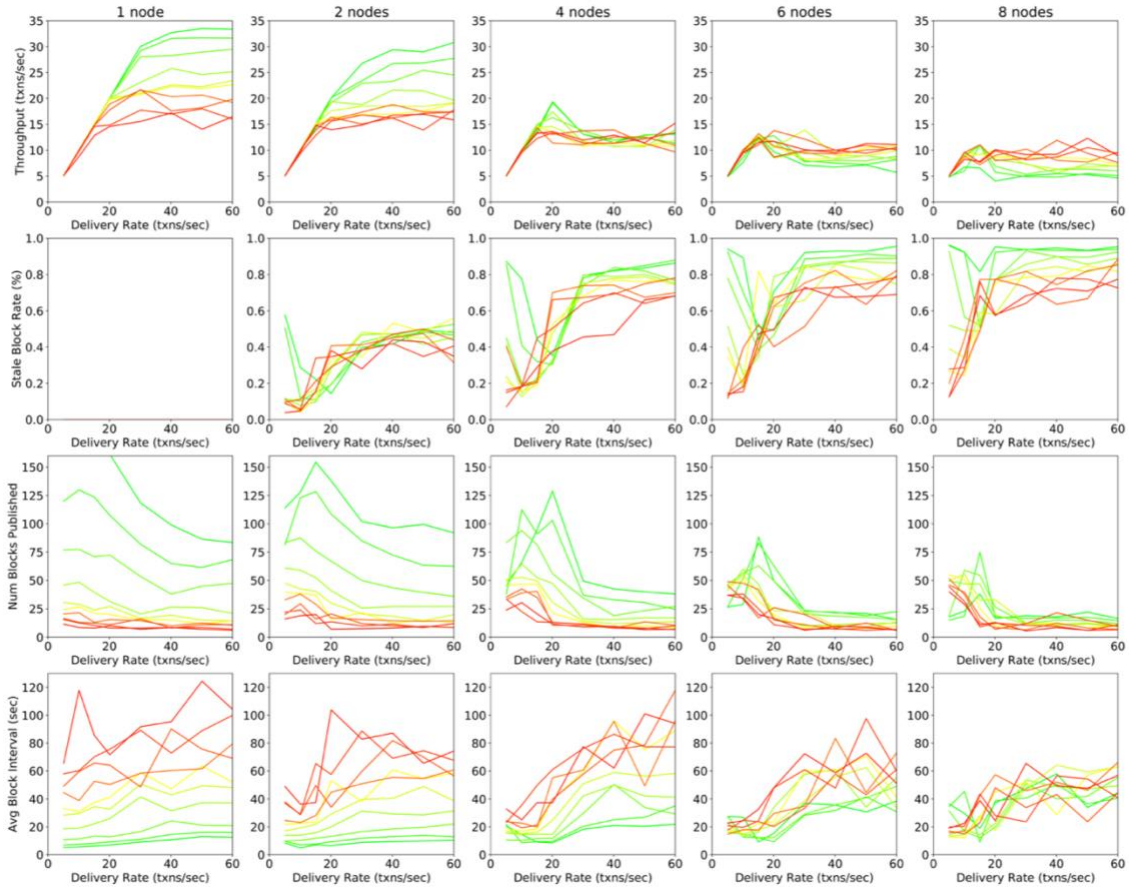

Figure 5: Average throughput, stale block rate, number of published blocks, and observed block interval (rows) for networks of size 1, 2, 4, 6, and 8 (columns). Each metric is plotted as a function of transaction delivery rate and target block interval (color, see legend).

The choice of block interval has several implications. As discussed in section 2.8, the block interval and the block size together establish an upper bound on the possible throughput of the network. The shorter the block interval, the higher the throughput bound will be. However, a high theoretical upper bound isn't necessarily achievable in a real network environment. For example, at a 20-second block interval with 2,000 transactions/block (the parameters used in Figure 4), the theoretical upper throughput bound is 100 tps. The experimental results fall far short of this throughput even in a single-node network, which can process fewer than 35 tps.

In a multi-node network, lengthening the block interval is likely to increase observed throughput even while the theoretical bound is reduced. In practice, the throughput of a network suffering from stale block overhead will be optimized at the choice of block interval where empirical throughput and the theoretical upper bound meet. Empirical throughput may plateau if constrained by computing resources (as observed in Figures 4 and 5), in which case choosing the shortest block interval that achieves the plateau value will have the additional benefit of decreasing transaction latency. Depending on the application supported by the blockchain, transaction latency may not matter, or it may be more important than the throughput itself, in which case an adequately short block interval may be necessary regardless of its performance implications.

3. Discussion

Computing resources proved to be a key limitation to the conclusions that can be drawn from the performance of the simulations. The relationship between throughput and network size did follow the same trend as the results in [10], with throughput declining as the network grows. However, it is possible that this decline was affected not only by the overhead of stale blocks, but also by increased competition for computing resources in larger simulations. Evidence that the throughput decline is due to more than resource competition is given by the stale block rates observed while varying the block interval. The very different effect of raising the block interval on the 1- and 2- node networks (a decline in throughput) compared with the 6- and 8-node networks (an increase in throughput) indicates that the prevalence of stale blocks becomes the primary performance bottleneck as the network grows.

CHAPTER VI

FUTURE WORK

1. Experimental design

While the observed trends in this data do inform performance considerations, PoET is designed to be scalable to many more than 8 nodes. The purpose of conducting performance research is precisely to design networks that perform well on a large scale. Given the limited computing resources on which these networks were simulated, even 6- and 8- node networks experience precipitous performance declines at very low transaction rates, making it difficult to understand their behavior with high resolution. A next step will be to perform the same analyses on a computing cluster, in which each node can be given far more computing resources and kept isolated from its peers. This will allow for a direct comparison in the behavior of the networks while varying only the factor of computing resources, as well as for new simulations at much larger network sizes.

The experimental setup can be improved in several ways. Most importantly, each experimental run should last as long as is feasible for the most accurate results. A suggestion of 100 published blocks was made by the Sawtooth developers to get an accurate estimate of the real throughput, which, dependent on observed block intervals, can fluctuate greatly if the amount of data is too small. At 20 second block intervals, this would mean experimental durations of approximately 20 seconds/block * 100 blocks = 2000 seconds (about double the 800 – 1000 seconds used in these experiments). At longer block intervals, the necessary duration to publish 100 blocks would increase proportionally. The Sawtooth developers also suggested waiting until at least 50 blocks have been published before beginning to collect performance data. 50 blocks is the default value of the parameter that determines when PoET begins fully relying on historical data to parameterize the wait time distributions. In theory, setting initial_wait_time equal to target_wait_time * network_size should yield the same behavior both before and after this threshold, but this equivalence should be experimentally verified.

2. Propagation delay and block size

The other key factor that affects the stale block rate is the propagation delay among network nodes. This delay can depend on the size of the blocks being propagated, the propagation policy of the consensus protocol, the topology of the network, and the underlying infrastructure of the internet. The size of blocks is a primary factor in propagation delay, and also one of the few that can be easily and dynamically modified. Larger block sizes mean more transactions can be incorporated in a single block (which would tend to increase the throughput), but also that each block takes longer to propagate and validate (which would tend to increase the likelihood of stale blocks and decrease throughput). In theory, the ideal block size can be larger if the block intervals are longer, and should be smaller in the case of shorter intervals. It remains unclear whether one of these parameters might dwarf the other in importance, and what their combined effects might be. Experimenting with variations in block size in a simulated network would also require enforcing realistic propagation delays that might occur in a real network.

The effects of block size may also depend on implementation-specific behavior of a particular transaction family. For example, the amount of time required to validate a large block of IntKey transactions may be significantly shorter than the amount of time required to validate a block of work-intensive transactions with complicated dependencies. Therefore, determining ideal block sizes may depend on the interaction of several factors, including block interval, network size, network bandwidth, computing resources, and the specific transaction family.

In addition to the size of blocks, the network topology in Sawtooth can be explicitly designed, or left up to the protocol to maintain. A node's peering behavior can be specified as *static* or *dynamic*. Static peering requires each node to specify an explicit list of peers upon startup. Dynamic peering requires only one or more seeds, and the node automatically explores the network topology to forge new connections. Dynamic peering allows specification of the minimum and maximum number of peers for that node. This data uses dynamic peering and a minimum connectivity of 1. In larger networks, connective topology may affect the block propagation delay, and in turn the likelihood of stale blocks.

3. Improving PoET and Sawtooth

While PoET has potential to serve as a fully public, permissionless consensus protocol, there are still several vulnerabilities that need to be addressed. The first is its susceptibility to Sybil attacks, in which a single attacker forges multiple node identities to place a larger proportion of the network under her control. In traditional PoW, the literal expensiveness of participating in the consensus protocol disincentivizes attacks of this nature – an attacker can forge as many identities as she wants, but the cost of computing is the same whether those computations are performed at one node or at several. In PoET, controlling multiple node identities is indeed advantageous, because each node receives its own wait time during each round. While the specialized SGX hardware required to participate in PoET may serve as some barrier to extensive Sybil attacks, this is a one-time, fixed cost that is likely to shrink as the technology matures. PoET's statistical z-test defense is only capable of detecting a single fraudulent node, but cannot detect whether a group of nodes is controlled by the same party.

Another concern in public networks are Denial-of-Service (DoS) attacks, in which an attacker floods the network with requests that tie up computing resources and impede normal functionality. The Bitcoin network disincentivizes this with transaction fees, and has very little computational capability to exploit. Ethereum, with its Turing-complete smart contracts vulnerable to computational abuse, uses the concept of "gas" to meter a transaction's execution. A transaction in Ethereum can only invoke the network to compute on its behalf to the extent that it has pre-paid for the work. Sawtooth's transaction processors, being equally expressive and free to implement arbitrarily compute-intensive work, need a mechanism to protect themselves from malicious transactions that might seek only to tie up validators' resources. Currently, any such protection must be custom-implemented in the transaction processor itself, and may not lend itself to the flexibility required in a truly open network.

4. Formal modeling

The absolute results from empirical studies may not translate well to novel network scenarios. It is not the absolute throughput values that are valuable, which depend on many instance-specific parameters such as the underlying computing hardware, but rather the relationships among key system parameters. With more robust empirical data in a resource-rich computing environment, the creation of formal mathematical models will be a valuable contribution. For example, the combination of block interval, propagation delay, and network size determine the stale block rate, but the precise relationship among these variables has not been determined. This relationship is likely to be implementation-specific and therefore may differ between, for example, a Sawtooth network and the Bitcoin network. Therefore, protocol-specific modeling may be required. The stale block rate, in turn, determines the expected decline in performance from an unhindered single node. The stale block rate is also a crucial factor that affects the security of a blockchain. The creation of these models will make it possible to understand how fixing or bounding top-priority aspects of the system influence or determine other components. For example, if a particular use-case requires a certain minimum transaction latency, this will force a maximum choice of block interval, and determine the expected throughput and security capability of the system.

CHAPTER VII

CONCLUSIONS

This thesis is the first work that investigates the implications of directly varying the block interval in conjunction with the network size on empirical throughput in PoET consensus. Considering that the choice of block interval determines both the theoretical bound on throughput and gives rise to the observed throughput, the optimal value of this parameter (assuming throughput is of primary concern) will occur at the point that the theoretical and empirical values meet. This point, however, depends on additional factors such as the size of the network, the processing power of its nodes, and the block size (or propagation latency). Given the difficulty of creating meaningful performance analyses in complex distributed systems, this work establishes a foundation of environmental considerations and parameters for conducting future empirical analyses and constructing formal relational models. A better quantitative understanding of the interactions among myriad system parameters will be crucial for efficiently optimizing real world blockchain networks and facilitating industry adoption.

A. EXAMPLE HYPERLEDGER CALIPER BENCHMARK CONFIGURATION FILE

```
---
test:
  name: intkey
  description: IntKey workload for Caliper-> Sawtooth
  clients:
    type: local
    number: 1
  rounds:

  - label: 5TPS5sec
    txDuration:
    - 40
    rateControl:
    - type: fixed-rate
      opts:
        tps: 5
    arguments:
      txnPerBatch: 20
    callback: benchmark/intkey/intkey_set.js


monitor:
  type: docker
  docker:
```

B. DOCKER COMPOSE FILE FOR SINGLE NODE SAWTOOTH NETWORK

```
# 1_compose_poet1.0.yaml


version: '2.1'

services:

  shell:
    image: hyperledger/sawtooth-all:1.0
    container_name: sawtooth-shell-default
    depends_on:
      - rest-api-0
    entrypoint: "bash -c ¥"¥
        sawtooth keygen && ¥
        tail -f /dev/null ¥
        ¥""

  validator-0:
    image: hyperledger/sawtooth-validator:1.0
    container_name: sawtooth-validator-0
    expose:
      - 4004
      - 8800
    ports:
      - '4004:4004'
      - '8800:8800'
    command: |
      bash -c "
        if [ -z $$(ls -A /var/lib/sawtooth) ]; then
          sawtooth keygen && ¥
          sawadm keygen --force ¥

          sawset genesis ¥
            -k /etc/sawtooth/keys/validator.priv ¥
            -o config-genesis.batch && ¥
          sawset proposal create ¥
            -k /etc/sawtooth/keys/validator.priv ¥
            -o config.batch ¥
            sawtooth.consensus.algorithm=poet ¥
            sawtooth.poet.report_public_key_pem=¥"$$(cat /etc/sawtooth/simulator_rk_pub.pem)¥" ¥
            sawtooth.poet.valid_enclave_measurements=$$(poet enclave measurement) ¥
            sawtooth.poet.valid_enclave_basenames=$$(poet enclave basename) ¥

          poet registration create ¥
            -k /etc/sawtooth/keys/validator.priv ¥
            -o poet.batch && ¥
          sawset proposal create ¥
            -k /etc/sawtooth/keys/validator.priv ¥
               sawtooth.poet.target_wait_time=20 ¥
               sawtooth.poet.initial_wait_time=20 ¥
               sawtooth.poet.block_claim_delay=1 ¥
               sawtooth.poet.key_block_claim_limit=100000 ¥
               sawtooth.poet.ztest_minimum_win_count=999999999 ¥
               sawtooth.publisher.max_batches_per_block=100 ¥
            -o poet-settings.batch && ¥

          sawadm genesis config-genesis.batch config.batch poet.batch poet-settings.batch
        fi;

        sawtooth-validator -v ¥
          --endpoint tcp://validator-0:8800 ¥
          --bind component:tcp://eth0:4004 ¥
          --bind network:tcp://eth0:8800 ¥
          --peering dynamic ¥
          --network trust ¥
          --minimum-peer-connectivity  1
      "
    environment:
      PYTHONPATH: "/project/sawtooth-core/consensus/poet/common:¥
        /project/sawtooth-core/consensus/poet/simulator:¥
        /project/sawtooth-core/consensus/poet/core "

  poet-validator-registry-tp-0:
    image: hyperledger/sawtooth-poet-validator-registry-tp:latest
    container_name: sawtooth-poet-validator-registry-tp-0
    expose:
      - 4004
    depends_on:
      - validator-0
    command: poet-validator-registry-tp -C tcp://validator-0:4004
    environment:
      PYTHONPATH: /project/sawtooth-core/consensus/poet/common
    stop_signal: SIGKILL

  rest-api-0:
    image: hyperledger/sawtooth-rest-api:1.0
    container_name: sawtooth-rest-api-0
    expose:
      - 8008
    ports:
      - '8008:8008'
    depends_on:
      - validator-0
    entrypoint: sawtooth-rest-api -vv -C tcp://validator-0:4004 --bind rest-api-0:8008

```

REFERENCES CITED

[1] M. Herlihy, "Blockchains from a Distributed Computing Perspective," *Communications of the ACM,* vol. 62, no. 2, pp. 78-85, 2019.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system.," December 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf. [Accessed 2018].

[3] A. Antonopoulos, Mastering Bitcoin: Programming the Open Blockchain, Beijing: O'Reilly, 2017.

[4] "Bitcoin Historical Price & Events," [Online]. Available: https://99bitcoins.com/price-chart-history/. [Accessed 2019].

[5] A. Antonopoulos and G. Wood, Mastering Ethereum: Building Smart Contracts and Dapps, Beijing: O'Reilly Media, 2018.

[6] "CrytoKitties," [Online]. Available: https://www.cryptokitties.co/. [Accessed 2019].

[7] "Hyperledger. Blockchain technologies for business," The Linux Foundation, [Online]. Available: https://www.hyperledger.org. [Accessed 2019].

[8] "The Linux Foundation," 2018. [Online]. Available: https://www.linuxfoundation.org/press-release/2018/07/hyperledger-passes-250-members-with-addition-of-9-organizations/.

[9] K. Kshetri, "Blockchain's roles in meeting key supply chain management objectives," *International Journal of Management,* pp. 80-89, 2018.

[10] H. Dang, A. Dinh, E. C. Chang and B. C. Ooi, "Chain of Trust: Can Trusted Hardware Help Scaling Blockchains?," *CoRR,* 2018.

[11] S. Bano, A. Sonnino, M. Al-Bassam, S. Azouvi, P. McCorry, S. Meiklejohn and G. Danezis, "SoK: Consensus in the Age of Blockchains," University College London, UK and The Alan Turing Institute, 2017.

[12] A. Gervais, K. Wüst and H. Ritzdorf, "On the Security and Performance of Proof of Work Blockchains," *IACR Crytol. ePrint Arch.,* 2016.

[13] K. Zhang and H. A. Jacobsen, "Towards Dependable, Scalable, and Pervasive Distributed Ledgers with Blockchains," *Middleware Systems Research Group, University of Toronto,* 2018.

[14] W. Li, A. Sforzin, S. Fedorov and G. Karame, "Towards Scalable and Private Industrial Blockchains," *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts,* 2017.

[15] A. Baliga, "Understanding Blockchain Consensus Models," Persistent Systems Ltd., 2017.

[16] C. Cachin and M. Vukolić, "Blockchain Consensus Protocols in the Wild," *IBM Research Zurich,* 2017.

[17] B. Liskov and M. Castro, "Practical Byzantine Fault Tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, New Orleans, ISA, 1999.

[18] J. Fischer, A. Lynch and S. Paterson, "Impossibility of Distributed Consensus," *ACM,* vol. 32, no. 2, pp. 374-382, 1985.

[19] "Bitcoin energy consumption," [Online]. Available: https://digiconomist.net/bitcoin-energy-consumption. [Accessed December 2018].

[20] Q. Nasir, I. A. Qassse, M. A. Talib and A. Nassif, "Performance Analysis of Hyperledger Fabric Platforms," 2017.

[21] H. P. a. S. W. Group, "Hyperledger Blockchain Performance Metrics," 2018.

[22] A. Dinh, J. Wang, G. Chen, R. Liu and B. Ooi, "BLOCKBENCH: A Framework for Analyzing Private Blockchains," *ACM,* pp. 1085-1100, 2017.

[23] L. Xu, N. Shah, Z. Gao, Y. Lu and W. Shi, "On Security Analysis of Proof-of-Elapsed-Time (PoET)," *Springer International Publications,* pp. 282-297, 2017.

[24] "BlockBench: private blockchains benchmarking," [Online]. Available: https://github.com/ooibc88/blockbench. [Accessed 2018].

[25] M. P. Caro, M. Ali, M. Vecchio and R. Giaffreda, "Blockchain-based Traceability in Agri-Food Supply Chain Management," *IEEE,* pp. 3-6, 2018.

[26] M. Hulea, O. Rosu, R. Miron and A. Astilean, "Pharmaceutical Cold Chain Management: Platform based on a distributed ledger," *IEEE International Conference on Automatic Quality Testing,* vol. 6, pp. 1 - 6, 2018.

[27] "Hyperledger Sawtooth Documentation 1.0.5," The Hyperledger Project, 2018. [Online]. Available: https://sawtooth.hyperledger.org/docs/core/releases/1.0/contents.html. [Accessed 2019].

[28] "PoET Specification," The Hyperledger Project, [Online]. Available: https://sawtooth.hyperledger.org/docs/core/releases/latest/architecture/poet.html. [Accessed 2019].

[29] "Ripple," [Online]. Available: https://ripple.com. [Accessed 2018].

[30] D. Schwartz, N. Youngs and A. Britto, "The Ripple Protocol Consensus Algorithm," Ripple Labs Inc. White Paper, 2014.

[31] "Stellar," [Online]. Available: https://stellar.org.

[32] D. K. Tosh, S. Shetty, X. Liang, C. Kamhoua and L. Njilla, "Consensus Protocols for Blockchain-based Data Provenance: Challenges and Opportunities," *IEEE,* pp. 469-474, 2017.

[33] A. Wahab and W. Memood, "Survey of Consensus Protocols," 2018.

[34] D. Mazieres, "The Stellar Consensus Protocol: A federated model for Internet-level consensus," 2016. [Online]. Available: https://www.stellar.org/papers/stellar-consensus-protocol.pdf.

[35] M. Milutinovic, W. He, H. Wu and M. Kanwal, "Proof of Luck: an Efficient Blockchain Consensus Protocol," in *Conference Proceedings SysTEX*, 2016.

[36] J. Alwen, G. Fuchsbauer, P. Gazi, S. Park and K. Pietrzak, "Spacecoin: A Cryptocurrency Based on Proofs of Space," *IACR Cryptol. ePrint Arch.,* pp. 1 - 26, 2015.

[37] S. Dziembowski, S. Faust, V. Kolmogorov and K. Pietrzak, "Proofs of Space," 2015.