LIVENESS ANALYSIS, MODELING, AND SIMULATION OF BLOCKCHAIN CONSENSUS ALGORITHMS' ABILITY TO TOLERATE MALICIOUS MINERS

By

Amani Altarawneh

Anthony Skjellum
Professor of Computer Science
University of Tennessee
at Chattanooga
(Chair)

Richard R Brooks
Professor of Electrical &
Computer Engineering
Clemson University
(Committee Member)

Lu Yu
Assistant Professor of Electrical &
Computer Engineering
Clemson University
(Committee Member)

Farah Kandah
Associate Professor Computer Science
University of Tennessee
at Chattanooga
(Committee Member)

LIVENESS ANALYSIS, MODELING, AND SIMULATION OF BLOCKCHAIN CONSENSUS ALGORITHMS' ABILITY TO TOLERATE MALICIOUS MINERS

By

Amani Altarawneh

A Dissertation Submitted to the Faculty of
the University of Tennessee at Chattanooga in Partial
Fulfillment of the Requirements of the Doctor of Philosophy in Computer Science

The University of Tennessee at Chattanooga
Chattanooga, Tennessee

August 2021

ABSTRACT

The blockchain technology revolution and concomitant use of blockchains in various applications have resulted in many organizations and individuals developing and customizing their own fit-for-purpose consensus algorithms. Because security and performance are principally achieved through the chosen consensus algorithm, the reliability and security of these algorithms must be both assured and tested. This work provides a methodology to assess such algorithms for their security level and performance is required; liveness for permissioned blockchain systems is evaluated. We focus on permissioned blockchains because they retain the structure and benefits afforded by the blockchain concept while end users maintain control over their processes, procedures, and data. Thus, end users benefit from blockchain technology without compromising data security. We expect that this methodology and taxonomy can be applied to other types of blockchains.

The developed methodology is used to provide a liveness analysis of byzantine consensus algorithms for permissioned blockchains. We provide a Digital Ledger Technologies (DLTs) consensus algorithm classification to understand the miner-selection process. We compile the "security ingredients" that enable consensus algorithms to achieve liveness, safety, and byzantine fault tolerance (BFT) in blockchain systems. We organize these requirements as a new taxonomy that describes requirements for security. And, Brewer's theorem is utilized to explain tradeoffs between availability and consistency in consensus algorithm design. This analysis uses formal methods and techniques and is applied to two exemplary consensus algorithms: lightweight mining (LWM) and byzantine fault-tolerant Raft (Tangaroa). Our analysis reveals the liveness of the given consensus algorithm and its ability to protect against malicious miner denial of services (DoS) attacks. Digital signatures are employed to prove integrity and non-repudiation of messages passing in the systems. Queueing theory and Markov chains are applied to determine the average waiting time of client transactions when malicious miners work to slow the system. Queuing theory and Markov chains jointly are employed to test a given blockchain's ability to perform correctly despite the presence of malicious miners or resistant nodes. Overall, the methodology presented here provides a roadmap to guide developers during the design phase of consensus algorithms to render these algorithms more secure and robust.

DEDICATION

To my dear parents, Mohammad and Qutnah Altarawneh, who instilled in me the value of education very early in life and who stood by me, gave me endless support, trust, and love, and encouraged me every step of the way to be here today. I am ever so thankful and appreciative to you, Mum and Dad.

To my family, especially my brother Moaz, for setting the example for me and inspiring me to pursue my dreams as I saw him pursuing his PhD in the US.

ACKNOWLEDGMENTS

First of all, I thank God for all the blessings given to me in this life. Thank you to my parents as well as my brothers and my sisters for giving me the encouragement and energy to finish this long journey.

I want to express my sincere appreciation to my advisor, Dr. Anthony Skjellum, for all his support throughout my PhD journey. Dr. Skjellum, thank you for your generosity in guiding me and mentoring me with patience and kindness, your insightful support made my PhD experience rich, fulfilling, and great. Your support made it possible for me to complete my dissertation, I could not have done it without you. I am fortunate to have had the opportunity to learn from you and I look forward to continue collaborating with you.

I would like to thank Dr. Farah Kandah and Dr. Lu Yu for serving on my committee. I appreciate the time you took to read my dissertation. Your comments and suggestions improved the quality of this dissertation. Special thanks to Dr. Richard Brooks for his invaluable support, input, comments, and guidance. It has been an honor working with all of you and I look forward to continue collaborating with you.

To all SimCenter friends and staff, especially Kim Sapp, for the care and love she provided to me since day one. And, finally, to my family and friends, thank you for the prayers and for loving me through it all.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

GLOSSARY

- **Blockchain—** is a distributed ledger that is controlled by multiple untrusted nodes. It is a chain of records that is continuously growing by adding a new block every time the nodes reach an agreement on a new block of data [3].
- **Consensus Algorithm—**is a process to achieve agreement on value(s) among the nodes in distributed systems [33], or the process that ensure an agreement is reached on each block that is appended to the ledger.
- **Miner—**Miner is a node that creates blocks of transactions from time to time and adds each in turn to the distributed ledger [85].
- **Mining—**is the process where a miner creates a block of transactions and adds it to the the blockchain [85].
- **Byzantine Fault Tolerance—**is the property of a system that can continue operating even if some of the nodes fail or act maliciously [98].
- **Byzantine Consensus—**is an algorithm allows nodes to operate correctly despite the presence of less than third of malicious participants (e.g., miners) [3].

- **Denial of Service (DoS) Attack—**is a threat that breaches the security of a system. It occurs when a malicious miner makes it impossible for honest miners to get the chance to work or for legitimate clients to access the system [85].

- **Availability against DoS Attacks—** is the ability of consensus algorithm (miners running the consensus) to progress correctly despite the presence of malicious miners, where they work to slow the system or drop transactions [85].

- **Termination—**for a consensus algorithm, the behavior that nodes eventually form a consensus on a value or values [33].

- **Liveness—**is a property of a system that eventually allows "good things to happen." In other words, it refers to a progress where an action or event is eventually executed despite the presences of attackers [72]. Liveness of a blockchain consensus algorithm is measured with two properties: the ability of the consensus algorithm to terminate, and the system availability against malicious miner DoS attack [3].

- **Faults vs. Failures vs. Errors—**failure occurs when the system cannot provide the expected service because of fault or faults (i.e., when a fault or faults become active); any incorrect step in a process is a fault, an error is an abstract or concrete measure of the deviation from expected behavior resulting from a failure [58].

- **Safety—**is a property of a system that doesn't allow "bad things to happen." In other words it is a property of a system where ERROR state is not reached [72]. Safety of a blockchain is measured with the ability of nodes to reach an agreement on who mines the next block, the integrity of the transactions and on the consistency of the chains for majority of nodes, and the total order of the transactions.

- **Agreement—**for a consensus algorithm, is the state in which all nodes agree (form consensus) on a given value or values. Agreement means that all honest nodes should accept the submitted transaction and add it to the blockchain or else discard it [33].

- **Double Spending—**is an attack where the attacker can duplicate or falsify certain digital information [63].

- **Integrity—**means consistency of all accepted transactions in all honest nodes (which means no double spending) [33].

- **Total Order—**which is the sequence of blocks and transactions in a correct chronological order [33].

- **Consistency—**means all nodes in the system maintain the same copy of the blockchain in the same order of blocks. This refers to the safety concept, which stops "bad things from happening" [3].

- **Node Authority—**is the ability of a given node to read, write, and/or commit transactions in the blockchain [3].

- **Synchronicity of the Network**—the network settings that determine how nodes communicate with each other by sending messages bidirectionally [33].
- **State**—is a snapshot of the current status of the system. Any change in the current variables is considered a system transition to a different state. Examples of system states follow: new transaction(s) are submitted, one miner is offline/busy, a new block is created and broadcast, etc [2].
- **State-Change-Based Classification**–is a way to categorize consensus algorithms based on how state-changes are decided [2].

CHAPTER 1

INTRODUCTION

This chapter outlines the importance of consensus algorithms to a blockchain's performance as well as this dissertation's motivation to establish rigorous theoretical methods to assess blockchain consensus algorithms components (liveness and safety). The dissertation aims to develop a roadmap for blockchain practitioners to test and evaluate the liveness and safety specifically of preselected-leader consensus algorithms for permissioned blockchain. The importance of blockchain protocols and systems to critical institutions and infrastructures of modern society makes it both timely and important to understand the strengths, gaps, and benefits of blockchain-based distributed ledgers.

1.1 Background

Many of the institutions upon which modern society depends—hospitals, banks, stock exchanges, airports, and so on—need highly reliable and available infrastructure to function properly. Infrastructure systems, however, often experience single points of failure in some parts of their architecture, which makes them susceptible to crashes, downtime, and/or hacking. Therefore, distributed systems are designed with a network of connected machines and distribute the work and resources across this network to perform tasks and to protect against single points of failure. For such distributed systems to function properly, all participating machines/entities/actors must be able to agree on key aspects of the current state of the system; otherwise, the system would not be able to work in unison as a coherent group and any benefits associated with distributing the system would largely be lost. Further, the distributed system must be able to withstand the failure of a subset of its members to be sufficiently resilient and be highly available. Distributed systems must also be safeguarded against insider threats arising from the circumstances of machine compromises. The process of agreeing on the current state of a group or deciding what is the next state is called *reaching consensus* [50], which, in the case of distributed systems, is typically carried out via consensus algorithms.

Blockchain is a single decentralized distributed data ledger, that is replicated and shared among multiple distributed nodes, in which transactions are grouped in blocks in a chronological order [33]. Blockchain technology has recently been employed in many applications other than its original application, cryptocurrencies. There is a large body of research that proposes various solutions using such blockchain technology. However, many of these solutions are yet to be adopted, in part because of security and performance limitations [33]. Blockchain protocols are designed to establish trust among participants that need not trust each other yet must work together to reach agreement on what should be entered in the events chain. Blockchain technology enables system decentralization in which there is no single authority or group of members that may collude to gain control or to alter distributed ledger data (e.g., the permissionless blockchain Bitcoin [83]). To reach consensus, each blockchain's consensus algorithm must be able to guide the process of validating the current blockchain, help decide which node should create the next state (i.e., the next block), assist in validating the next state, and ensure that all nodes agree on the next state [96]. An effective consensus algorithm must carry out these operations in an objective and mathematically rigorous manner to guarantee that all nodes agree on the state of the system while preventing the system from failing because of faulty or malicious behavior [33]. A blockchain is permissioned if membership is required to join.

In this dissertation, we focus on permissioned blockchains because they are resistant to Sybil attack (A Sybil attack occurs when one actor creates multiple identities and participates in the consensus as different independent nodes), which has a harmful impact on consensus output [38]. And, permissioned blockchains keep the structure and benefits afforded by the blockchain idea while the end-users (companies and businesses) maintain control over their processes, procedures, and data. Thus, businesses benefit from blockchain data management without compromising sensitive data security.

This dissertation will assess and evaluate consensus algorithms against a subset of threats that occur because of the malicious behavior of the nodes/miners, these threats will be explained in Chapter 5. Such an evaluation will comprise a significant contribution to the literature in blockchain protocols and distributed systems.

1.2 Motivation

The first implementation and application of blockchain distributed ledger technology was Bitcoin, which uses proof of work (PoW) as its mining algorithm [84]. PoW is resource-intensive [54] in order to limit inflation of the underlying cryptocurrency. However, non-economic applications of blockchain technology do not need to limit the number of mined blocks, so using PoW would be both wasteful and exorbitant in unrelated areas such as data-provenance and other distributed consensus applications [105]. In fact, blockchain technology has also been employed recently in many applications other than its original cryptocurrencies application. Because of Bitcoin's success, there are many industries seeking to utilize blockchain's technological potential in applications other than cryptocurrency. There is also a large body of research and studies that propose various technical solutions using blockchain technology [33]. We assert that these have yet to be adopted, in part, because of security and performance limitations [33]. Blockchain provides security by having miners with conflicting interests work together. This type of protocol works to remove incentives for collusion and can guarantee data immutability for all participants. But, we also need to guarantee that miners cannot abuse their presence in the system to favor certain parties. In this context, a feasible permissioned system that includes conflicts can still be operated successfully.

Thus, for the best utilization of blockchain technology, the main components that have the most impact on system security vulnerabilities and what tradeoffs emerge between functionality (e.g., scalability, performance) and security must be studied. A core component of blockchain technology is the consensus algorithm; thus, any actions that hinder participants from reaching consensus are chief concerns when considering security [113]. Designing and evaluating the consensus algorithm is challenging because of the wide range of conflicting factors involved in achieving consensus. Lamport defined a correctly distributed system as a system that has both liveness and safety properties [72, 75]. In other words, a system that allows “good things to happen” and “prevent bad things from happening.” Additionally, the system should be byzantine fault tolerant, which keeps the system functioning in an adversarial environment. Therefore, a definition of a secure consensus algorithm would be a correct and byzantine fault-tolerant algorithm [72, 75].

This raises the need to study the properties and factors that make the consensus algorithm both secure and reliable. Many companies and programmers are developing and customizing consensus algorithms to suit their requirements. These algorithms must be secure to be able to achieve their goal. Startup company developers and industry programmers alike rely on their ideas, knowledge, and experience to establish trust in their developed protocols and not on formal methods or established techniques [33].

Blockchain-based systems work but there is no theoretical understanding as to why they work. Thus, a formal methodology based on a rigorous approach that helps with evaluating and assessing the security of the consensus algorithm for blockchain systems is required. Established techniques must be applied to analyze consensus algorithm behavior and that algorithm’s ability to counteract known attacks. Particularly, a roadmap needs to be provided to test and evaluate the level of liveness and safety of consensus algorithms and their resilience to relevant attacks, such as malicious miner denial of service (DoS) attacks.

1.3 Objectives

The widespread interest in using blockchain technology for non-cryptocurrency applications has led to developing many blockchain consensus algorithms (which form the core of every blockchain protocol/system). The main goal of this work is to devise a way to measure to what extent should distributed ledger consensus protocols be trusted as both secure and safe. This work will guide developers and programmers to achieve this trust by validating liveness and safety properties and prove protocols’ resilience against threats in an adversarial environment. It is not enough for the programmers and developers to use their experience to design a secure consensus protocol, they also need to apply mathematical models, formal methods, and other tests to verify all possible scenarios resulting from the presence of malicious behavior. Using formal methods reveals protocol vulnerability gaps and uncovers scenarios that might cause system halts, effective impacts from insider threats, and/or ambiguous states. Another goal for this work is therefore to provide the research community with an assessment and feedback on whether the applied formal methods yielded positive results while providing recommendations as to which methods lead to definitive findings at the end. The combination of developers' experience, formal methods, and human judgment will offer the best assessment of whether a consensus algorithm is secure or not for a specific application or applications.

1.4 Dissertation Statement

Given the motivations offered above, a methodology is needed to assess and evaluate the security of consensus algorithms, particularly liveness and safety, based on formal methods and established techniques. This dissertation achieves these goals as follows:

1. Provide a common consensus algorithm classification based on how state changes are decided, focusing on the miner-selection process.
2. Identify "security ingredients" and map them to determine the ingredients that enable a given consensus algorithm to achieve liveness, safety, and byzantine fault-tolerance (BFT).
3. Using the Brewer's Theorem (The CAP theorem), analyze the tradeoffs between liveness and safety in terms of availability and consistency, respectively, in consensus algorithms design.
4. Evaluate the preselected-leader algorithms (LightWeight Mining (LWM) [4] and BFT-Raft [39]) for liveness (availability) against malicious miner DoS attacks using established techniques and formal methods such as Markov chain and queueing theory.

1.5 Research Questions

Research questions for this work are defined as follow:

- **RQ-1**: Which consensus algorithm classification(s) (if any exist) is/are better for security evaluation (liveness and safety)?
- **RQ-2**: What are the security requirements that enable a given consensus algorithm for blockchain systems to achieve liveness, safety, and byzantine fault-tolerance (BFT)?
- **RQ-3**: Do the preselected-leader consensus algorithms prioritize both consistency and availability according to the CAP Theorem tradeoffs?
- **RQ-4**: What formal methods are used to test the liveness (availability) of leader-based consensus algorithms (LWM and BFT-Raft) for a permissioned blockchain? Which work and which do not?
- **RQ-5**: What is the impact of the number of malicious miners on the liveness (availability) of the blockchain system? How is the system resilient to the malicious miner-based Denial of Service (DoS) attack?

1.6 Contributions

The main contributions of this dissertation will be a roadmap to test and evaluate the liveness and safety of preselected-leader consensus algorithms for permissioned blockchain. This roadmap provides a centralized evaluation methodology that includes a consensus algorithm classification scheme, a security ingredients' scheme (a new contribution), and a new availability analysis for two of the preselected-leader consensus algorithms based on Brewer's Theorem (The CAP Theorem) and Lamport properties (liveness and safety) for correct consensus algorithms.

These contributions are to be achieved via the following outcomes/deliverables/results:

- Provide a classification scheme that can be used to categorize consensus algorithms from several different types of distributed ledgers based on how the state-changes are decided. It is valuable to categorize each consensus algorithm so that they can be compared and contrasted to identify each category's strengths and weaknesses [2].

- The “security ingredients” that enable any consensus algorithm to achieve liveness, safety, and byzantine fault tolerance (BFT) are then studied in both permissioned and permissionless blockchain systems. Particularly, the security requirements of preselected-leader consensus algorithms for permissioned blockchain are determined.

- The CAP Theorem is utilized to explain important tradeoffs in consensus algorithms design between liveness and safety which are respectively availability and consistency.

- Formal methods are applied to evaluate and test the consensus algorithms for the security requirements. Security analyses for two byzantine consensus algorithms—LightWeight Mining and BFT Raft—are then provided as representative of efficacy and applicability of these methods.

The major impact of this work will be to provide developers with a technically valid roadmap to find the optimized security requirements of the consensus algorithm for a respective blockchain variety during the early development process and to incorporate these tradeoffs and features into the basis of their specific designs. Also, this work will provide the means to test and evaluate existing consensus algorithms that are already in use for liveness and safety. Finally, developers, researchers, and users of blockchain systems will be able to know the strengths, weaknesses, and areas of potential enhancement for any given system.

1.7 Evaluation and Metrics

Success of this dissertation will be measured as follows:

- To what extent has each of the research questions been answered satisfactorily? And, in particular, do the availability analysis results against malicious miners denial of service DoS attacks work for either or both LWM and BFT Raft.

- Is the work able to deliver the evaluative capabilities proposed for blockchains with regard to liveness and safety?

- Is there evidence that the techniques developed can be generalized, through additional work, to analogous and more general blockchain protocols?

The experimental plan and experiments already performed (and published) will drive the determination for answering the research questions and satisfying the overarching goal of making a significant contribution to evaluating liveness and safety for permissioned blockchain protocols.

1.8 Audience

The audience of this dissertation work is blockchain practitioners such as developers, experts, and researchers of blockchain systems who seek to develop and/or investigate secure consensus algorithms. Startup company developers and industry programmers who want to combine their experiences with formal methods or established techniques to achieve secure and robustness consensus algorithms, and researchers who are investigating which formal methods work (and which do not) for evaluating and assessing the security and robustness.

1.9 Outline

The remainder of this dissertation is organized as follows: Chapter 2 provides background information on necessary topics relevant to this work and the related work. Chapter 3 provides a new consensus algorithms classification based on how the system state-changes are decided, focusing on the miner-selection and mining processes. The organization of blockchain consensus algorithm security requirements and a new taxonomy that describes the requirements for security are described in Chapter 4. Chapter 5 illustrates the methodology/approach and describes the experimental set-up, simulations to collect results. Chapter 6 describes the results and their analysis and presents methods that did not work in the analysis. We conclude the dissertation in Chapter 7 by discussing our findings and then outline our future work in Chapter 8.

CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

This chapter presents the background for different types of consensus algorithms in distributed and blockchain systems and provides the literature review of other research regarding the evaluation of the security of the consensus algorithms of permissioned/permissionless blockchain systems[1].

Here, we differentiate distributed ledger technology (DLT) from blockchains and directed acyclic graphs (DAGs). We explain blockchain data validity functions and techniques that are used to certify the data stored in a blockchain. We explain the main blockchain security properties and indicate blockchain types based on the ability of miners to join the given network. We illustrate distributed consensus algorithm properties in terms of consistency, availability, and partition tolerance (CAP). We explain the origin of consensus in distributed systems, and we explain two byzantine consensus algorithms: the Lightweight Mining Algorithm (LWM) and BFT-Raft (Tangaroa). Finally, we present a description of related work in consensus algorithm classification and evaluations methodologies.

2.1 Distributed Ledger Technology (DLT)

It is important to differentiate distributed ledger technology (DLT) from blockchains and directed acyclic graphs (DAGs). A DLT is a set of data that is replicated, shared, synchronized, and spread across multiple sites globally with no central administration or centralized data storage [11]. DLTs have several desirable properties such as immutability, transparency, anonymity, trust, and decentralization [8, 78]. Immutability means that once a transaction exists within the ledger, it cannot be changed. DLTs are transparent because they provide an open record of transactions

[1]The remainder of this dissertation will focus on permissioned blockchain algorithms/protocols, but this chapter is more general in its review and discussions.

without any intermediates or third parties, and all the participants in the network have access to the information that is recorded in the ledger. While the identities of the participants in the network in DLTs are anonymous, DLTs offer a platform that can enable trusted communication between parties that may not necessarily trust each other [8]. Additionally, DLTs have a single source of truth where the data is edited in one place only [78], so all participants have the same copy of the ledger. Finally, most DLTs are decentralized geographically but are owned and maintained by a central authority [11].

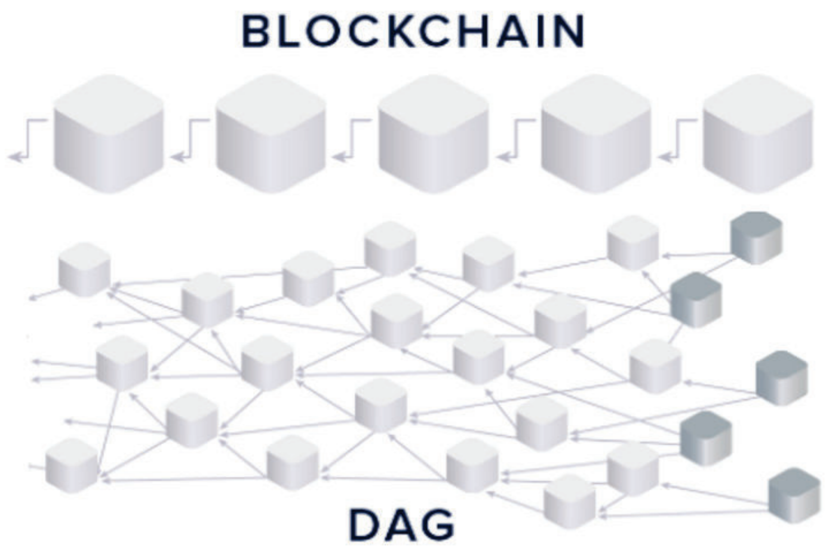


Figure 2.1 Distributed Ledger Technologies (DLTs) Types, Blockchain vs. Directed Acyclic Graphs (DAGs) (adopted from [61])

Blockchains and DAGs are both types of DLTs, each with its own unique characteristics [53]. A blockchain is a data structure in which transactions are grouped together into larger structures called blocks [87]. Each block is created using information about the previous block, which links the two blocks together. This ensures that all the data is immutable; if the information in one block is changed, it will not match what is stored in the block immediately after it, and the discrepancy will be easily detected [116]. A DAG is a data structure that is similar to a blockchain, but with some key differences. While blockchains have a single chain of blocks, DAGs allow for several chains to exist and connect simultaneously as shown in Figure 2.1. To achieve this behavior, an individual transaction is added to the chains as it is validated, as opposed to grouping transactions together into blocks and validating the block as a whole. Because transactions are added as they are validated and transactions are validated by individual nodes, DAGs have a tree-like structure in which each transaction has an edge connecting it to another transaction that validates it. Despite their differences, blockchains and DAGs both rely on consensus algorithms to ensure that all nodes agree on the state of the system [19].

2.2 Blockchains and Consensus Algorithms

Blockchain Technology refers to a class of system where data is recorded in a way that makes it difficult or impossible for attackers to change or fake the stored data. As shown in Figure 2.2, blockchain technology is built or designed based on a peer-to-peer (P2P) network topology, where nodes or peers are equal, and they are able to function as servers or clients to other nodes in the network at the same time [33, 98]. Also, blockchain is known as a distributed ledger technology (DLT) where the data stored in it is shared, synchronized, and replicated globally on thousands of machines by miners [2]. And, a miner is a node/machine/server that creates blocks of transactions from time to time and adds each in turn to the distributed ledger; this process is called mining. Blockchain improves efficiency and builds trust for entities transacting with one another [33, 98].

The origin of blockchain is in distributed systems where multiple trusted nodes can agree on a common truth [102]. Blockchain is however a distributed ledger that is controlled by multiple untrusted nodes. And, it is a chain of records that is continuously growing by adding a new block every time nodes in the network reach an agreement on the processed data. Blockchain acts as a trusted system because the stored data is replicated over all the nodes and a hash of the previous block is included in the current block. A consensus algorithm ensures agreement is reached on each block where it is appended to the ledger with no trust in a single machine or authority (e.g., public blockchain) [86]. Blockchain might fork where two blocks are added to the chain at the same time because of a network split and miners become misaligned. This result into different version of the blockchain [23].

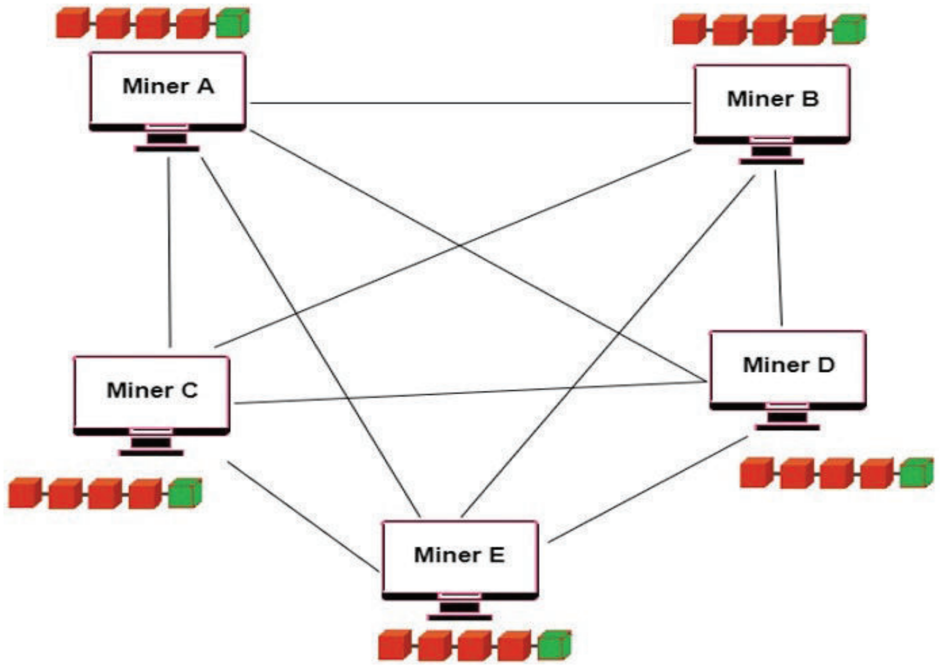


Figure 2.2 Blockchain Technology Peer-to-peer Network Model

2.2.1 Blockchain Data Validity Functions & Techniques

The validation of blocks and the data (transactions) stored in the blockchain is achieved using various functions and techniques; These functions and techniques are hash pointer, cryptographic hashing, asymmetric key, and Merkle trees [17, 52], see Figure 2.3 illustrate the Bitcoin block data. In blockchain design, hash pointer provides the security backbone of the chain. In each block, there is a hash pointer points to the next block with its hash. Thus, any altering or tampering with chain would be easily detected by the hash value. These hashes must be cryptographic hashes for sufficient assurance, as discussed next.

Cryptographic hashing is a mathematical method that is used to convert any kind of text in any size into a string of a fixed size of characters. This hashing method is collision resistant; thus, it is computationally expensive to create the same output for two distinct inputs, and the input cannot be calculated using the output. Another cryptographic technique used is the use of asymmetric keys (asymmetric cryptography), which is when a pair of keys that are linked together. The sender encrypts the message using one key using the receiver's public key while the receiver decrypts the message using its key (which is private). The digital signature is a combination of the asymmetric keys plus hashing cryptography, and transactions submitted to blockchain are required to be signed by the miners using their digital signature[2] [51]. Finally, a Merkle tree is data structure that where each transaction is hashed, then each pair of transactions are combined together in one hash, this continues until there is one hash for the entire block called the Merkle root. This Merkle root is used to quickly verify if any of the transactions under consideration are already included in another block [85].

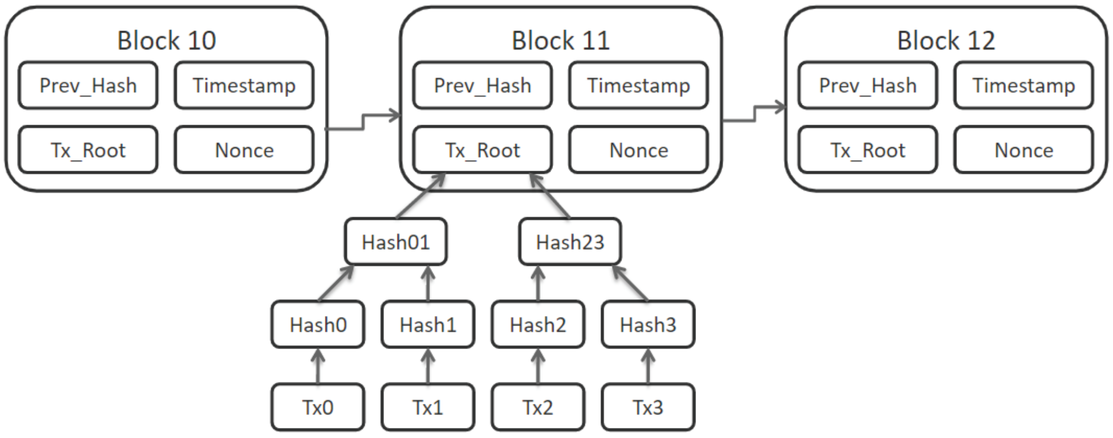


Figure 2.3 Blockchain Technology (Bitcoin Block Data) Adopted from [22]

2.2.2 *Blockchain Security Properties*

Blockchain has several desirable security properties by its design that makes blockchain establishes trust among different parties. These properties are achieved using the functions and techniques described in Subsection 2.2.1. The security properties are data integrity and non-repudiation, blockchain immutability, and consensus on each block. We explain them here:

- Data integrity in a blockchain is guaranteed when any corruption or tampering of the data stored in blocks or in the exchanged messages between the miners can always be detected [21].

[2] A digital signature applies to an instance of some plaintext or crypto text that is to be signed. Digital signature means that the sender encrypts a cryptographic hash of data first with the receiver's public key, then the sender's private key. The recipient decrypts first with the sender's public key and then with the receiver's own, private key. This provides the hash in clear text, which can be compared to the hash of the data received and thus establishes the source (for non-repudiation) as well as the intended recipient, plus the data's integrity and completeness. The digital signature is orthogonal from the confidentiality of the data transferred itself.

- Non-repudiation in blockchain is guaranteed because neither the sender nor the receiver of a message can deny transmitting or receiving the messages. Digital signatures make deniability impossible; they provide high confidence authentication.

- Immutability means the data kept in a blockchain ledger cannot be altered, and remain unaltered and unchanged, blockchain uses cryptographic or a hash value to achieve that– without any such alternations being detectable.

- Consensus in distributed systems is a state in which all nodes agree on the same value(s). This value is more trusted when majority of nodes/miners in the network agreed on it. Consensus in blockchain technology refers to the miners ability to agree and verify the validity of the all transactions in each block, that involve reaching a consensus on the miner to mine the next block.

2.2.3 Permissionless vs. Permissioned Blockchain

Blockchain systems are classified based on the authority that is given to the nodes to access the data (*read, write, and commit*) and to participate in the consensus of the appended blocks. They are also classified based on the type of node membership (i.e., public to everyone or private and limited to a set of trusted nodes). Organizations design their blockchain according to the node membership type and data access authority [14]. A blockchain is *permissionless* if anyone can *read, write, and commit*, where nodes are neither trusted nor known and membership is not required. Bitcoin is an example of permissionless blockchain [83]. A blockchain is *permissioned* if a membership to join is required. Permissioned blockchains fall into one of three categories (public , consortium, and private) according to the right to access the data. Permissioned blockchain is *public* if any node in the network have the right to *read*. However, the right to *write* is reserved only for the trusted nodes (members), and *commit* is limited to a subset of trusted nodes (members). Hyperledger fabric is an example of permissioned blockchain [31]. A *consortium* blockchain limits the right to *read* in the public permissioned blockchain only to the trusted nodes (members). Corda [29] is an example of a consortium blockchain. A *private* blockchain limits the right to *read* to the trusted nodes (members), and the right to *write*, and *commit* is limited only to the network

Table 2.1 Nodes Membership - Blockchain Technology Classification (adopted from [42])

	Permissionless	**Permissioned**		
Access	**Public**	**Public**	**Consortium**	**Private**
Read	Anyone	Anyone	Members	Members
Write	Anyone	Members	Members	Administrator
Commit	Anyone	Anyone/Members	Anyone/Members	Administrator

administrator. See Table 2.1 for more details on the classifications of blockchain based on the authorization to access the blockchain. The authorities that are given to the nodes have a direct impact on the design of the consensus algorithms, where the vulnerabilities are gradually increased with the size and dynamicity of the set of nodes that are reaching consensus.

Some of popular consensus algorithms for permissionless blockchains are Bitcoin, which uses the Proof-of-Work (PoW), and Ripple [99], which uses the Ripple Consensus Protocol. Miners in the Bitcoin PoW algorithm need to find a value that solves a puzzle quickly to add to the blockchain and earn cryptocurrency as a reward. Solving the puzzle involves computing many hashes, which consumes computational power. Additionally, the PoW algorithm is vulnerable to 51% attacks, which can occur if one entity owns more than half of the total processing power in the Bitcoin network [114]. Similarly, the Ripple consensus protocol is designed so that each miner must create a list of other trusted miners that never collude against it. This list, called the Unique Node List (UNL) [14], must have less than 40% overlap with any other UNL to prevent collusion. Each miner selects a set of transactions and broadcasts them to the miners in its UNL, the UNL then validate and vote on the transactions. Voting happens in multiple rounds and only transactions with a super-majority 80% or more votes become a valid block.

Ethereum and Intel SawtoothLake [14] are blockchain platforms that are considered both permissioned and permissionless. In Proof-of-Stake (PoS), which Ethereum is said to be adopting soon [110], to prevent malicious behavior, miners are selected randomly with the ones with more stake having more likelihood of being selected. Hence, miners need to maintain a sufficiently large amount of cryptocurrency in order to have a higher chance to be selected to create the block and earn that new block's reward. It is worth mentioning here the risk for "wealthy" miners to

Table 2.2 Blockchain Consensus Algorithm Comparison

Consensus algorithms	**Permission to join**	**Energy cost**	**Ability to fork**	**Blockchain Technology**
PoW	permissionless	High	Yes	Bitcoin
PoS	both	Low	Yes	Ethereum soon
DPoS	permissionless	Low	Yes	Bitshares
Ripple Consensus	permissionless	Low	Yes	Ripple
PoET	both	Low	Yes	Intel SawtoothLake
PBFT	permissioned	Low	No	HyperLedger Fabric
Raft	permissioned	Low	No	Zookeeper [117]
LWM	permissioned	Low	No	Scrybe
BFT Raft	permissioned	Low	No	None

monopolize the process over the long run. Another variant of the Proof of Stake (PoS) consensus algorithm that is implemented in BitShares [24] is Delegated Proof of Stake (DPoS) [25]; DPoS is used in permissionless blockchain where participants stake an amount of cryptocurrency to qualify as candidates to mine the next block.

Intel SawtoothLake uses the Proof of Elapsed Time (PoET) consensus algorithm [14], which relies on specialized hardware. All miners must run a Trusted Execution Environment (TEE) which randomly generates waiting times for each miner. The miner with the shortest waiting time as generated by the TEE mines the next block. Practical Byzantine Fault Tolerance (PBFT) [36] is a consensus algorithm that is used in the permissioned blockchain HyperLedger Fabric [14]. PBFT uses the state machine replication concept [30] and requires $3n+1$ replicas to be able to tolerate n faulty nodes. Messages sent between nodes and replicas are signed and encrypted to decrease the number of messages needed to maintain trust. However, overhead in the network increases with the number of replicas. Raft is used in permissioned blockchain where the nodes collaborate with each other to elect a leader who produces the next block. Once a leader has been elected, another phase begins where the leader receives a transaction, adds it to its log, and broadcasts the log to all other nodes [88].

2.2.4 Correctness of Consensus Protocols

Lesile Lamport designed a scheme by which the correctness of a distributed system can formally be proven. Lamport stated that a system is correct if the following two criteria are true: a system that doesn't allow bad things to happen, and a system that eventually allows good things to happen which are safety and liveness, respectively [74]. However, safety and liveness properties are difficult to fulfill simultaneously. For any distributed system to be correct, it must come to some form of consensus on the correct answer. The correct consensus algorithm requirements are validity, agreement, and termination. Validity means that any value decides upon the network must be proposed by one of the processes. In other words, the consensus algorithm cannot arbitrarily agree on some result or be hardcoded to always return the same value. Agreement means that all non-faulty processes must agree on the same value, it is pointless to bother with the results of the faulty processes. In order for the result to be meaningful, all functioning processes must agree on it. The last requirement is termination, it means that all the non-faulty processes eventually will decide on the value. The system must eventually return some value, it can't wait forever for the value to be decided. So, both validity and agreement are safety properties, they specify things that can never happen in a system. Reaching consensus correctly by having validity and agreement ensures that honest nodes never decide on a trivial, random, or different values. Termination, on the other hand, is a liveness property as it specifies what must happen for a system to be considered correct. Without the guarantee of a termination there is no guarantee that consensus will ever be achieved [74].

2.2.5 Brewer's Theorem (The CAP Theorem)

CAP stands for the three components of the distributed system trilemma: consistency, availability, and partition tolerance [27]. These components are not disconnected from the safety and liveness properties. Consistency defined as every node providing the most recent state of the system, if it does not have the most recent state, it will not provide any response. This builds off the idea of safety since it is referring to something that will never happen. Consistency in a system is where no two nodes will return a different state at any given time, and no node will return an outdated state. Availability means every node has constant read and write access. An available system allows a user to make updates and retrieve states without delay. Availability promises what must happen, which is the ability to read and write some state, and that is liveness. Partition is the inability of two or more nodes in a network to communicate with each other. There is a partition in the network if node ‘A’ cannot receive messages from node ‘B’ and vice versa. Partition tolerance is the ability to function despite partitions within the network. Partition tolerance is more difficult to categorize, but it falls in line with safety, as it specifies what will not happen.

2.3 Byzantine Consensus algorithms

Byzantine consensus algorithm has the ability to function despite the presence of malicious miners, and a consensus algorithm byzantine fault tolerant, if it tolerates 50% + 1 of the total number of miners crashing or less than third of the total miners being malicious. However, the analysis provided in this dissertation studies the ability of the consensus algorithms to tolerate malicious miners when they work to slow or stop the system. In this dissertation, we cover two byzantine consensus algorithms Lightweight mining algorithm for Scrybe (LWM) and Bft-RAFT Tangarou and they are described in the following sections:

2.3.1 Lightweight Mining Algorithm for Scrybe (LWM)

Blockchains provide a data structure that can be used to guarantee integrity and non-repudiation of data, as well as maintaining provenance metadata for systems. The lightweight Mining (LWM) algorithm provides these guarantees with minimal resource requirements [4, 20]. The LWM algorithm departs from the resource-intensive (and time-consuming) verification approaches that cryptocurrencies, such as Bitcoin and Ethereum, use when expanding the blockchain [82].

Transactions and blocks are the main components in the Scrybe blockchain. Computational actors called miners are responsible for mining the blocks and adding them to the blockchain. Mining is the process of including a transaction in a data block and adding that block to the blockchain. A transaction in Scrybe includes client details such as name, signature, and public key, as well as the submission timestamp, the hash of the source data using the SHA-3 algorithm, and the persistent URLs (PURLs) that point to the source data (extraneous to the blockchain). Miners aggregate a group of transactions and add them as one block to the blockchain upon a successful mining step. Miners use the Merkle root [17] to quickly verify if any of the transactions under consideration are already included in another block. The LWM algorithm selects a miner that will be responsible for adding the block to the blockchain and broadcasting the block to other miners to prevent duplicate blocks. In addition to the Merkle root, miners use the hash of the previous block and the miner's signature to check for potential malicious behavior such as a miner purposely omitting transactions or unauthorized miner(s) broadcasting invalid blocks [55].

In LWM, Hash numbers broadcast by miners can be based on a miner-generated random number. Collusion occurs when several miners work to manipulate the process. However, LWM can function as long as good miners protect against collusion [90]. The LWM algorithm requires all miners to share their hashes first. This requirement prevents miners from withholding their number and be able to tamper with the process. Without this requirement, a miner could wait until all other miners have submitted their numbers, then generate a number that excludes or selects a certain miner. The enormous amount of time needed to invert (reverse-engineer) the hashes created by the LWM algorithm makes this situation impractical and highly unlikely to happen. Each miner must include its hash to protect against deniability. The LWM algorithm can continue functioning as long as there is at least one good miner (the $\sum_j r_j \mod N$ remains random). As long as the transaction falls into a good miner's pool, the transaction will be mined, but it will take longer for the transaction to be added to the blockchain if there is a large number of bad miners [28, 55].

2.3.2 Tangaroa and RAFT

RAFT stands for Reliable, Replicated, Redundant, and Fault-Tolerant. And since its creation by Leslie Lamport in 1998, the Paxos algorithm [71] has been the reference consensus algorithm that all others look to for inspiration. However, it is difficult to understand and implement, which led Diego Ongaro and John Ousterhout to develop Raft as a simple and practical alternative to Paxos [88]. In Raft, one node is the leader and the rest are followers, unless a new leader is in the process of being elected. The leader has finite amount of time they are allowed to lead, which is called the term. Each node has its own individual set of all of the transactions the system has received, which is called their log. When a new transaction is received, Raft collapses consensus into two phases: log propagation and leader election. Leader election occurs if there is no current leader, the leader is unavailable, or the current leader’s term has ended. Each node is assigned a random amount of time they must wait. The first node to wait their assigned amount of time becomes a candidate and votes for itself. That node then asks all other nodes to vote for it, which they do if they have not already cast a vote for the current term and the candidate’s log is either matches or is longer than their own. Once one node receives a vote from a majority of other nodes, that node is elected as the leader for the next term [88]. This approach guarantees that each node has an equal chance of being elected as the next leader. Once a leader has been elected, the log propagation phase begins. During this phase, the leader receives any transaction in the system, adds the new transactions to their log, and broadcasts their log to the followers. The followers then replicate the leader’s log to their own log. Once a majority of the followers responds confirming they have replicated the leader’s log, the leader commits the transaction and notifies the person who requested the transaction. If there is a discrepancy between the leader’s log and a follower’s log, the follower will send a rejection message. The leader must then go through each previous entry of their log and broadcast it to the followers until the discrepancy is resolved. Once the leader’s term ends, the leader election phase begins and the process repeats [88].

Byzantine nodes break Raft, Tangaroa (BFT Raft) [39] is intended to be an extension of the consensus algorithm Raft that tolerate Byzantine behavior. In Raft, any node can start the election process and terminate the current term, a malicious node can exploit that and starve the system with fake elections and badly impact the system availability. A malicious leader could modify the client’s request and violate its integrity or instruct replicas to commit entries without an agreement by all other nodes. Also, a malicious leader could confuse the clients by either ignoring their requests or responding with arbitrary results. Thus, there is a need for some modifications to Raft to handle such Byzantine nodes [33, 39].

BFT Raft has the following modifications to handle Byzantine behavior, note that (RPCs) stands for remote procedure calls :

- The leader in BFT Raft replicates the messages with the sender’s digital signature to prevent any forging that might occur by the leader and to verify the integrity of the messages [39].

- BFT Raft prevents malicious leaders from starving the system by allowing the clients to terminate the leadership for any leader in case of no progress [39].
- BFT Raft node uses a cryptographic hash to prove it has the entire log replicated and ready to append a new entry, which makes it easier for other nodes to quickly verify the node's log using the signature and the hash [39].
- Nodes could verify that the leader won the election by counting and validating the votes (RequestVoteResponses) [39].
- In BFT Raft, each AppendEntriesResponse RPC is broadcasted to each other node, rather than just to the leader. Thus the nodes commit by itself by tracking the received AppendEntriesResponse RPC from other nodes [39].
- Nodes vote for a candidate only if the current leader is faulty. This prevents any node from initiating elections unnecessarily and starve the system [39].

Tangaroa is a byzantine consensus algorithm that is best used for permissioned blockchain. This consensus algorithm should never violates safety because of its network model which is eventually synchronous where the messages are continuously delivered during the synchronous periods. The algorithm is live if the network model used is synchronous with timely message delivery and nodes are synchronized nodes. Since Tangaroa does not have a verification process, it might violate the liveness and safety criteria if a malicious node rushes its timeout to become the leader. The malicious node can refuse to work when it becomes a leader causing a violation of the liveness criteria. The leader node is responsible for transmitting the sequence to all other nodes, it may try to confuse other nodes on purpose by transmitting different sequence to different nodes causing some correct nodes not deliver the message which violates the safety criteria. Thus, it is not guaranteed in BFT Raft to achieve agreement [33, 39].

2.4 Related Work for Blockchain Consensus Classifications and Security Evaluations

This section discusses the related work in the area of consensus algorithms classification for distributed ledgers, the security requirements for correct and byzantine fault-tolerant blockchain consensus algorithms, and the consensus algorithms evaluation methodologies and assessing procedures using formal methods. Several classification schemes for both blockchains and consensus algorithms have been proposed before [34, 86, 98, 107]. We acknowledge the related work we came across, studied, and built upon in what follows. The consensus algorithms security evaluation related work which is divided into three areas: general evaluation, category evaluation, and Formal evaluation.

2.4.1 General Evaluation

Most of the work done in this area has mainly been in the area of blockchain consensus algorithms classifications, which are comparisons with respect to some of the security implications [106]. For example, certain studies survey consensus algorithms based on various factors such as the incentive mechanisms that encourage good behaviors and reward the miners for mining [107]. Others studied consensus algorithms from the classical byzantine fault-tolerance to Nakamoto consensus [83] based on the mining nature, node scalability, and transaction throughput and latency [18] while others reviewed consensus algorithm with regards to basic security properties and techniques that are used to evaluate them.

2.4.2 Category Evaluation

Category evaluation work covered security evaluation studies according to specific blockchain categories (e.g., permissioned and permissionless consensus algorithms, and cryptocurrency consensus algorithms). So, Cachin et al. [33] provided a study of consensus algorithms for permissioned blockchains and their correctness where they overviewed thirteen algorithms for liveness and safety [33]. Other authors covered the permissionless consensus algorithms and provided a framework to highlight the security and performance properties of the design when creating a block in the consensus algorithms [16]. Sankar et al. provide a comparative analysis (on Stellar consensus protocol, Corda, and Hyperledger) for blockchain consensus algorithms based on the node degree of centralization [98]. Nguyen and Kim proposed a binary classification [86] in which blockchain consensus algorithms are either proof-based where a miner proves, either through effort or chance, that they deserve to mine the next block. Or, vote-based consensus where the nodes communicate with each other before adding the proposed block to the blockchain. Quite recently, Ferdous et al. [49] surveyed top cryptocurrencies consensus algorithms from various categories and properties; they provided a taxonomy to analyze consensus algorithm suitability and provided a decision tree to select the right consensus algorithm considering security, energy consumption, and incentives. The aforementioned research does not sufficiently cover the security properties of a consensus algorithm with various blockchain types (e.g., permissioned, permissionless, etc.); that is, they restrict themselves to a survey of blockchains used in cryptocurrencies. Determining security properties plays a key role during consensus algorithm design and development, which provides the basics to design a methodology based on formal methods to analyze and evaluate the security of consensus algorithms.

2.4.3 Formal Evaluation

The formal evaluation area covers the evaluations that are done based on formal methods and established techniques. For example, Stefano De Angelis's work [6] used CAP theorem to assess security and performances of consensus algorithms for permissioned Blockchain. This work assessed which "two out of three" properties between consistency, availability, and partition tolerance can be assured at the same time for two proof of authority (PoA) algorithms. The analysis showed that PoW algorithms have weak consistency and high availability, where PBFT showed the opposite [6]. PoA algorithms are classified as voting based algorithms [2] (and the analysis of them is out of the scope of this proposal), but the analysis by using the CAP theorem is also part of our methodology.

Hambolu provided security analysis of a lightweight mining protocol for a blockchain named Scrybe which can be used by provenance systems to maintain trust. Hambolu used formal methods such as CSP, Petri nets [97], and Markov chains to find and analyze performance issues. Part of the safety and liveness analysis of LWM behavior and its ability to counteract known attacks has been done by Hambolu and Bhat [20, 57]. LWM liveness is addressed as the victim's transactions that eventually arrive at the good miners' pool [57], and it was proven using a Petri net. However, this proposal is interested in the availability analysis of LWM against malicious miners Denial of Service (DoS); which is the average waiting time before a client's transaction is added to the blockchain.

Another analysis for LWM liveness was done by Bhat [20]. Bhat leveraged existing Distributed Denial of Service(DDoS) attacks as a tool to evaluate LWM liveness. she checked its performance in presence of Transmission Control Protocol (TCP)- based flooding attacks such as SYN Flooding and its variants. And she used Lamport's byzantine generals' problem to set the bounds for the number of malicious miners that the algorithm can tolerate which is less than the third of the total number of miners to improve LWM safety [20].

2.5 Summary

In this chapter we presented the background for different types of consensus algorithms in distributed and blockchain systems. We differentiated distributed ledger technology (DLT) from blockchains and directed acyclic graphs (DAGs). We explained blockchain technology blocks and data validity functions and techniques. We explained the main blockchain security properties and the blockchain types based on the ability of miners to join the network. We Illustrated distributed consensus algorithm properties in terms of consistency, availability, and partition tolerance (CAP). We explained the origin of consensus in distributed systems, and we explained two byzantine consensus algorithms: Lightweight mining algorithm (LWM) and BFT-Raft Tangaroa. Finally, we presented a description of the related work in the consensus algorithm classification and evaluations methodologies.

CHAPTER 3

DIGITAL LEDGER TECHNOLOGY CONSENSUS ALGORITHM CLASSIFICATION

In this chapter, we provide a common consensus algorithm classification based on how the system state-changes are decided. We focus on the system state-changes during the mining process and the miner-selection process. Here, we provide a new classification for 10 Digital Ledger Technologies' (DLTs') consensus algorithms (for Blockchain and DAGs) into two categories; leader- and voting-based consensus algorithms.

3.1 Review of the Dissertation Statement and Research Questions

In this chapter, we provide the answer for the research question below:

- **RQ-1**: Which consensus algorithm classification(s) (if any exist) is/are better for security evaluation (liveness and safety)?

Here, we design a new classification of Digital Ledgers Technology (DLTs) consensus algorithms, and we explain why this classification is better to evaluate safety and liveness.

3.2 State-Change-Based Classification

Blockchains are a type of distributed system in which each node contains a state machine whose state must match all other nodes' states, without any node necessarily knowing or trusting the other nodes in the system. In order to reach consensus, each blockchain's consensus algorithm must be able to guide the process of validating the current blockchain, help decide which node (miner) should create the next block, assist in validating this block, and ensure that all nodes agree on that block [96]. An effective consensus algorithm must carry out these procedures in an objective and mathematically rigorous manner to guarantee that all nodes agree on the state of the system. In addition, the consensus should be achieved correctly despite the presence of a ratio of malicious behavior [33].

As discussed in chapter 2, there are already several ways of classifying both blockchains and consensus algorithms. However, there evidently is not a classification scheme that can be used to categorize consensus algorithms of blockchains as well as other kinds of distributed systems based on how the state-changes are decided. State-change based classification is a way to categorize consensus algorithms based on how the state-changes are decided. A state is a snapshot of the current status of the system and any change in the current variables is considered a system transition to a different state. Examples of system states follow: new transaction(s) are submitted, one miner is offline/busy, a new block is created and broadcasted, etc. Consensus algorithms were classified based on the state-changes because this approach lends itself to different scenarios which better analysis such as of Petri Net [97], communication sequential processes [59], queueing theory [69], and Markov modeling [79]. Also, it helps identify strengths and weaknesses of groups of similar algorithms instead of individual algorithms.

In this dissertation, we focus on system state-changes during the mining and the miner-selection process. There is a need to understand how the miner-selection process is happening or who is responsible for selecting a miner to mine the next block, because this leads to understand the consensus algorithm's ability to tolerate malicious miners. We need to know how random the miner-selection process would be since the miner-selection process should be fair. The randomness of miner-selection process protect against malicious miners actions and assures a fair chance for each miner to mine blocks.

Malicious miners' behavior during the mining process impacts the (liveness and safety)[1] of the consensus process (or the process of adding blocks to the blockchain). Malicious miners could drop transactions, refuse to mine when they get selected, hinder the rest of the nodes from reaching an agreement and split the network, etc.

[1]Chapter 4 considers the "security ingredients" that enable a given consensus algorithm to achieve safety and liveness.

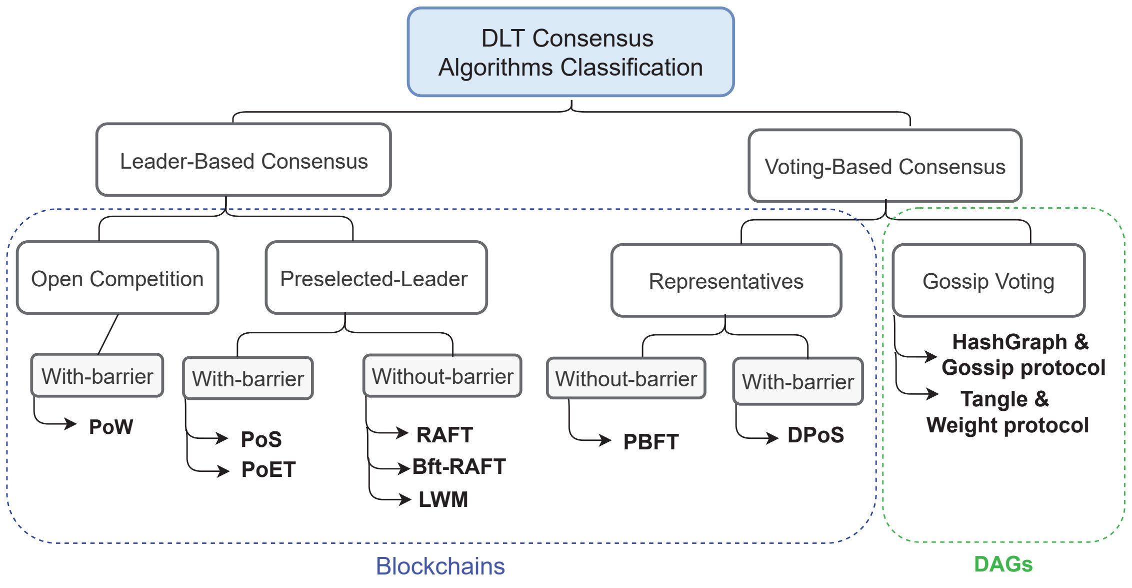


Figure 3.1 Distributed Ledger Technologies Consensus Algorithms Classification based on how the state-changes are decided, focusing on the miner-selection process

We outline two categories, leader-based consensus, and voting-based consensus, that can be used to describe and compare consensus algorithms from a variety of distributed systems. A new consensus-algorithm classification is presented in Figure 3.1[2]. Barriers mentioned in this figure are explained in the summary Table 3.1.

3.2.1 Leader-Based Consensus

In leader-based consensus algorithms, one node is chosen to be the leader. The leader is responsible for correctly ordering the transactions that are submitted into a block, broadcasting this block to all other nodes, and committing those transactions to the data structure. Though other nodes may validate the order that the leader provides, only the leader can change the order and number of transactions being committed. The leader can be selected in a variety of ways, but there is always at most one leader at any given time. Leader-based consensus algorithms are further classified into open competition and preselected-leader selection.

[2]Barriers are the cost that a miner has to give (e.g., pure power, specialized hardware, or capital investment) to join the network and create an account; barriers are used to protect against Sybil attacks.

3.2.1.1 Open Competition

In Open Competition selection consensus algorithms, miners compete for the right to become the next leader, who is responsible for compiling, ordering, and committing the next group of transactions, which usually provides a monetary reward. Typically, there is a task that each node is trying to complete, and the first node to complete the task is selected as the next leader. Regardless of if completing the task requires effort or luck, each node does whatever it can to increase its chances of completing the task and being selected as a leader. A famous example and the only consensus algorithm that fits this type is Proof of Work.

Proof of Work (PoW): PoW is the most prevalent consensus algorithm used by blockchains presently, mostly due to its use by Bitcoin [83]. Invented in 1993 by Cynthia Dwork and Moni Naor [46], PoW's simple design and rigorous security makes it an attractive solution for the consensus problem that all blockchains must solve. In PoW, nodes must generate a piece of data which satisfies given requirements that is computationally difficult to produce and easy to verify. Most PoW algorithms implement this by creating difficult puzzles, which can only be solved through brute force [83]. For example, Bitcoin's PoW algorithm, called Hashcash, requires each node to concatenate a given string with a number called a nonce. The node then hashes the resulting string and compares it to the current difficulty, which is the number that the hash must be smaller than in order to count as a correct answer. If the resulting hash is smaller than the difficulty, the node has found a correct answer. Otherwise, the node increments their nonce and repeats the process. Once a node finds a correct answer, they create a new block containing the nonce they used to calculate the answer, add the block to their copy of the blockchain, and broadcast their updated blockchain to all other nodes in the network. The other nodes see that a new block has been added, and add the new block to their copy of the blockchain once they calculate the hash using the winning nonce and verify that the resulting hash is lower than the difficulty. Every node that is currently mining Bitcoin is competing to find a nonce to concatenate to the hash of the previous block in the blockchain that hashes to a value lower than the current difficulty. Therefore, once a new block has been added, all mining nodes must restart the process of looking for a correct nonce because the hash they are concatenating the nonce with has changed [83].

3.2.1.2 Preselected-Leader Consensus algorithms

In preselected-leader selection consensus algorithms, miners also compete for the right to become the next leader, who is responsible for compiling, ordering, and committing a group of transactions, however, miners collaborate to elect their leader. In other words, miners agreed to give the right to mine the next block for one of the miners. The leader creates the block of transactions and broadcasts it to the other nodes in the system. The other miners add the block broadcasted by the leader and append it to their individual copy. The following are examples of preselected-leader selection algorithms:

Proof of Elapsed Time (PoET): One of the main criticisms of PoW is that, due to the difficulty of solving the puzzle, it consumes a lot of power without generating anything useful [105]. To address this issue, Intel created PoET [62] as an alternative to PoW that still selects its leaders competitively but does not waste large amounts of power to do so. Instead of solving a difficult puzzle, nodes in PoET are each given a random amount of time they must wait before creating a new block. Therefore, the node that is assigned the smallest amount of time will be allowed to create the next block and broadcast it to all other nodes in the network. To ensure that all random numbers are assigned fairly, Intel requires that each node has specialized hardware called the Software Guard Extensions (SGX). This hardware component ensures that the algorithm is running with trusted code, and can verify that the values it generates come from a protected environment.

By using verifiably random numbers, PoET ensures that each node has an equal chance of being selected as the next leader without requiring the nodes to waste energy generating solutions to puzzles. Because each node has an equal chance of being the next leader, there is nothing that any node can do to increase its chances of being selected, which differs from the meritocracy of PoW networks [62].

Proof of Stake (PoS): As blockchain technology evolved following the release of Bitcoin, it became obvious that PoW's scalability limitations and power consumption were too large for most applications [63]. To address these issues, Sunny King created PoS and implemented it in Peercoin [65], and PoS has since emerged as one of the main alternatives to PoW. In PoS, a node's likelihood to be selected to create the next block is proportional to the amount of the blockchain's cryptocurrency they own. This eliminates the need for nodes to complete complex puzzles, so PoS is more scalable and eco-friendly than PoW. Nodes in PoS algorithms compete economically rather than computationally, which raises its own set of problems that must be addressed to prevent malicious behavior. For example, certain safeguards must be put in place to ensure that the blockchain is not dominated by a small number of wealthy nodes [65]. Peercoin prevents this by basing a node's stake on their coin age, a number that considers both the amount of cryptocurrency the node has and the amount of time that node has owned the cryptocurrency. Coin age resets once a node is selected to create a block, which prevents wealthy nodes from dominating the block creation process. Additionally, while nodes may increase their likelihood of being selected by owning more coins, the leader selection has a random element to ensure that the selection process is fair [66].

Reliable, Replicated, Redundant, And Fault-Tolerant (RAFT) and BFT Raft (Tangaroa): In RAFT, one miner is the leader and the rest are followers unless a new leader is in the process of being elected. The leader has a finite amount of time they are allowed to lead, which is called the term. Each miner has its own individual set of all of the transactions the system has received, which is called their log. When a new transaction is received, RAFT collapses consensus into two phases: log propagation and leader election. Each miner is assigned a random amount of time they must wait. The first miner to wait their assigned amount of time becomes a candidate and votes for itself. That miner then asks all other nodes to vote for it, which they do if they have not already cast a vote for the current term and the candidate's log is either matches or is longer than their own. Once one miner receives a vote from a majority of other miners, that miner is elected as the leader for the next term [88]. Byzantine miners break Raft, so Tangaroa[39] is intended to be an extension of the consensus algorithm Raft that tolerates Byzantine behavior. BFT Raft (Tangaroa) follows the same process to select the leader as in RAFT. Both algorithms are described in Chapter 2.

Lightweight Mining (LWM): The LWM algorithm selects a miner that will be responsible for adding the block to the blockchain and broadcasting the block to other miners. The selected miner in each round is the leader for that round. In order to select a miner; the LWM algorithm requires all miners to generate secret numbers and each miner hashes its number then shares it with other miners. Share their hashes first prevents miners from withholding their number and be able to tamper with the process. Without this requirement, a miner could wait until all other miners have submitted their numbers, then generate a number that excludes or selects a certain miner. The enormous amount of time needed to invert (reverse-engineer) the hashes created by the LWM algorithm makes this situation impractical and highly unlikely to happen. Each miner must include its hash to protect against deniability [4]. The miner is selected based on the result of the sum of generated numbers *mod* the total number of miners.

3.2.2 Voting-Based Consensus

In voting-based consensus algorithms, nodes/miners vote for blocks that they think are valid directly, rather than voting for leaders or competing for the right to determine the order of the transactions. Or nodes/miners vote to select representatives to give the right to mine a block to one miner. Therefore, while the actual creation of the block may be done by one node/miner, the order, contents, and validity of the block are decided by multiple nodes/miners. This can happen synchronously, as with most blockchains, or asynchronously, as seen in DAGs.

3.2.2.1 Representative Voting

Voting based consensus algorithms that require each node to vote on the validity of every block are not typically practical and scalable in large distributed environments. However, a group of representatives may be elected as representatives responsible for proposing blocks. To maintain a certain degree of decentralization, the following conditions must apply:

1. All nodes must be eligible to be elected as representatives
2. All active nodes must participate in the election procedure
3. The election procedure must be conducted periodically to avoid centralization.

Based on this definition, we identify the following well-known consensus algorithms that fit this category.

Delegated Proof of Stake (DPoS): The Delegated Proof of Stake algorithm was proposed and developed by Dan Larimer, who implemented it in the EOS blockchain [25]. Slightly different versions of DPoS are also implemented in BitShares [24] and Steemit [76]. It is variant of the Proof of Stake (PoS) consensus algorithm, in which participants stake an amount of cryptocurrency in order to qualify as a candidate to produce the next block.

The DPoS consensus algorithm creates a technological democracy among [25] the participating nodes by creating a group of block producers [76], who are designated with the authority to propose new blocks. Users of the underlying cryptocurrency with an account can register as voters [10] to become stakeholders and vote for desired block producers. The algorithm functions in two phases: election of block producers and scheduling production of blocks [76]. The election process is straightforward. Registered voters may login to the voting portal and vote for a specific number of nodes, which is typically higher than the fixed number of block producers. For example, EOS has 21 block producers at any given time and stakeholders may vote for as many as 30 nodes in a session [10]. The 21 nodes with the highest number of votes are elected as block producers. They are then responsible for scheduling the production of blocks and receive a reward for their services.

Practical Byzantine Fault Tolerance (PBFT): In PBFT implemented distributed systems, nodes do not require a leader in the network and they can communicate with each other. However, one node is considered as the primary and the others are regarded as replicas. This property ensures that there are no forks in the global state of ledger. PBFT implements the state machine approach where transactions lead to state transitions, a client must wait for $f+1$ nodes to reply with the same resulting transition to regard a transaction as successful. In case the primary node fails to respond, nodes communicate with each other to decide which node will replace as primary and reach an agreement asynchronously [37].

The non-changing position of the primary node makes the PBFT algorithm extremely fast in comparison with other consensus algorithms. PBFT also requires $O(n^k)$ [60] communications between nodes where n is number of messages and k is the number of nodes. This increases exponentially with addition of new nodes to the network, which is not scalable across large network of nodes. Further, PBFT is susceptible to Sybil attacks [60] where one entity controls many different nodes with different identities. The possibility of carrying out a Sybil attack becomes almost negligible in large distributed systems with hundreds of nodes. Leaders are selected in a round-robin (the next one in line) manner and other nodes vote for the prospective leader's legitimacy to be the next leader.

3.2.2.2 Gossip Voting

Unlike pure voting and representative voting, gossip voting is a voting mechanism with little communication overhead in the network. This type of voting does not require leaders, followers, or competitions to reach consensus. Instead, it uses the gossip protocol, where every node randomly picks a neighbor node and sends all the information it knows to this node. This process repeats until, eventually, all the nodes in the network will have the same information. Additionally, every node will know where every other node got their information from. For example, imagine there are three nodes: Node 1, Node 2, and Node 3. If Node 1 and Node 2 are neighbors and Node 2 and Node 3 are neighbors, Node 3 will eventually know everything that Node 1 knows as well as the fact that Node 2 got their information from Node 1. This historical record of the spread of information ensures a fair ordering of transactions in the same way a blockchain does without requiring consensus to be reached synchronously. Thus, there is no need to have an explicit vote in order to reach consensus. Instead, each node votes that a transaction is valid once it sends the information about the transaction and where it came from to another node.

Therefore, the number of votes a transaction has is the number of times it has been sent from one node to another. All other nodes and it needs to confirm the received messages. In the gossip voting nodes don't have to send any messages, nodes can deduce what other nodes voted because they already send what they know.

Due to gossip voting's asynchronous and non-linear design, it is not suited for use with a blockchain data structure. Instead, this type of voting is done using DAGs. DAGs allow several different chains of transactions to exist simultaneously, which works well with gossip voting's tree-like structure. Despite their differences, though, both DAGs and blockchains are immutable, so there are not many security sacrifices associated with using a DAG as opposed to using a blockchain [1, 13]. The following are examples of gossip voting consensus algorithms, all of which are implemented using DAGs:

Swirlds Hashgraph: Developed by Leemon Baird in 2016 [13], Hashgraph was one of the first gossip voting algorithms to be implemented using a DAG. In Hashgraph, nodes send transactions randomly to other nodes in the network. Every time a node receives a transaction from another node, they create an event that records everything in the transaction. Additionally, the hash of both nodes' last event is recorded in the new event along with any new information that the receiving node wants to include along with the receiving node's digital signature and the timestamp. Over time, this builds a ledger of events in which events are linked by the hashes included when the event was created. Each node has a copy of this ledger, which may differ slightly from other nodes' ledgers if new gossip has not reached it, but after a certain point all the ledgers will be identical. Because of this, consensus can be determined by any node in the network; all nodes know almost everything that all other nodes know, so they know a node would vote if they were asked. This allows each node to be able to simulate elections through what is called virtual voting. It is called this because there is no communication happening during the voting process; it is all simulated by the individual node. This drastically reduces latency and communication overhead, which is one of Hashgraph's greatest strengths [13].

Tangle: IOTA [1] uses a similar consensus approach to Hashgraph with its gossip voting protocol Tangle [92]. IOTA developers implemented Tangle in 2015 to provide cryptocurrency for the internet of things systems without requiring competition or high overhead. In IOTA, transactions are called sites and links between two transactions are called edges. A transaction in Tangle must have edges to at least two other transactions in order to considered confirmed; if they have less than two edges, they are unconfirmed transactions, also called tips of the Tangle. When a new transaction is added, a Markov chain Monte Carlo algorithm is used to randomly select two or more tips of the Tangle to attach the new transaction to [92].

This means that every added transaction confirms two other transactions, which makes IOTA's Tangle scalable; the network does not slow down when there are a lot of new transactions. Tangle currently uses the help of PoW to reach consensus. When a node creates a new transaction, it does a small amount of PoW to determine the transaction's weight. When the transaction is added and becomes a site, its weight represents how trustworthy it is. As the site gets confirmed by more sites, it's the confirming site's weight is added to the confirmed site's weight. Thus, the more a site is confirmed, the more trustworthy it will be [1].

A summary of the consensus algorithms that have been classified above is provided in the Table 3.1 below. As mentioned earlier, in this dissertation we focus on permissioned blockchains and selected the Lightweight Mining and BFT-Raft as our subject study.

3.3 Summary

We provided a new consensus algorithms classification based on how the system state-changes are decided, especially the miner-selection and the mining processes. State-change based classification is a new classification reduced to practice in this dissertation. It is presented for 10 Digital Ledger Technologies' (DLTs') consensus algorithms (Blockchain and Directed Acyclic Graphs (DAGs)) and it classifies DLTs' into two categories: leader- based and voting-based consensus algorithm. Leader-based consensus algorithms are further sub-classified into two types: open competition and preselected-leader, where in both there is one miner that is responsible to create and broadcast the block. Voting-based consensus algorithms are further sub-classified into two types: representatives and Gossip voting. Representative-based consensus algorithms are either committee or Practical Byzantine Fault (PBFT) algorithms. This classification provides the answer for the research question **RQ-1**, as shown in Table 3.1 and meet research statement **#1**.

Table 3.1 Distributed Ledger Technologies (DLTs) Consensus Algorithm Classification

DLT Consensus algorithm Classification										
Who control the State-changes	**Voting-based:** Nodes/Miners vote for blocks that they think are valid directly, rather than voting for leaders or competing for the right to determine the order of the transactions. Or nodes/miners vote to select representatives to give the right to mine a block to one miner. Therefore, while the actual creation of the block may be done by one node/miner, the order, contents, and validity of the block are decided by multiple nodes/miners.				**Leader-based:** one miner is the leader, and it is responsible for correctly ordering the transactions that are submitted into a block, broadcasting this block to all other nodes, and committing those transactions to the data structure. And there is always at most one leader at any given time. and it is classified into categories: Open competition and preselected leader.					
DLT Type	**DAGs**		**Blockchain**							
Mining process	**Gossip-Voting (No miner selected):** This type of voting does not require leaders, followers, or competitions to reach consensus. Instead, it uses the gossip protocol, where every node randomly picks a neighbor node and sends all the information it knows to this node.		**Representative Voting:** A group of representatives (miners) may be elected as a committee of representatives who are responsible for proposing blocks. Or, miners vote for the leader legitimacy to be the next leader.		**Preselected leader:** Majority of miners agreed to give the right to mine the next block for one of the miners. The leader creates the block of transactions and broadcasts it to the other nodes in the system. The other miners add the block broadcasted by the leader and append it to their individual copy.					**Open Competition:** Open for any one to join, and the first miner to complete and solve the puzzle is selected as the leader.
Consensus Algorithm	HashGraph & Gossip protocol	Tangle & Weight protocol	Delegated Proof of Stake (DPoS)	Practical Byzantine Fault Tolerance (PBFT)	Proof of Stake (PoS)	Lightweight Mining (LWM)	BFT-Raft (Tangaroa)	Reliable, Replicated, Redundant, And Fault-Tolerant (RAFT)	Proof of Elapsed Time (PoET)	Proof of Work (PoW)
Energy Cost	Low	Low	Low	Low	Low	Low	Low	Low	Low	High
Permission to join	Permissioned	Permissionless or Permissioned	Permissionless	Permissioned	Permissionless or Permissioned	Permissionless or Permissioned	Permissioned	Permissioned	Permissionless or Permissioned	Permissionless
Miner-Selection procedure	No miner selection process; nodes send transactions randomly to other nodes in the network. Every time a node receives a transaction from another node, they create an event that records everything in the transaction.	No miner selection process; transactions are called sites and links between two transactions are called edges. A transaction in Tangle must have edges to at least two other transactions in order to considered confirmed; if they have less than two edges, they are unconfirmed transactions, also called tips of the Tangle.	Registered voters vote for a specific number of nodes, nodes with the highest number of votes are elected as block producers. They are then responsible for scheduling the production of blocks and receive a reward for their services.	Leaders are selected in a round-robin (the next one in line) manner and other nodes vote for its legitimacy.	A node's likelihood to be selected to create the next block is proportional to the amount of the blockchain cryptocurrency they own, and the amount of time that node has owned the cryptocurrency.	LWM requiring all miners to share their hash of secret values first. Then miners reveal and share their secret values, a miner is selected based on the result of the sum of generated numbers mod the total number of miners.	Each node is assigned a random amount of time they must wait. The first node to wait their assigned amount of time becomes a candidate and votes for itself. That node then asks all other nodes to vote for it, which they do if they have not already cast a vote for the current term and the candidate's log is either matches or is longer than their own.		The node that is assigned the smallest amount of time will be allowed to create the next block and broadcast it to all other nodes in the network	Once a node finds a correct answer, they create a new block containing the nonce they used to calculate the answer, add the block to their copy of the blockchain, and broadcast their updated blockchain to all other nodes in the network.
Barrier	No Barrier required		**Capital investment:** Candidates pool their money to increase chance of being elected.	No Barrier required	**Capital investment:** Candidate selection is directly related to stake size and age.	No Barrier required			**Specialized hardware:** Intel requires that each miner has specialized hardware called the Software Guard Extensions (SGX). Miners are each given a random amount of time they must wait before creating a new block.	**Pure Power:** Miners need to have powerful computers to solve difficult puzzles, which can only be solved through brute force.

CHAPTER 4

BLOCKCHAIN AND CONSENSUS ALGORITHM SECURITY

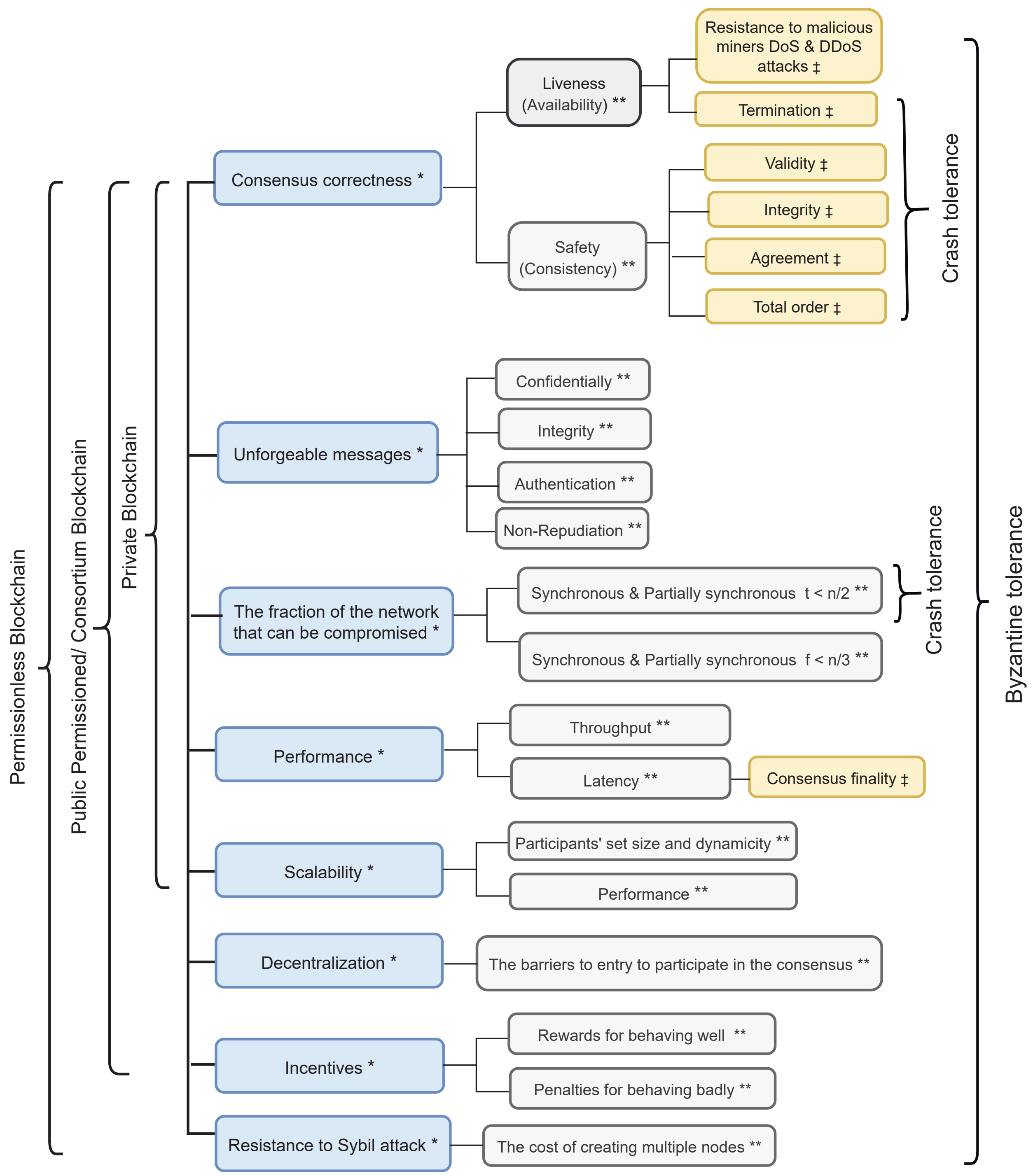
This chapter studies the “security ingredients” that enable a given consensus algorithm to achieve safety, liveness, and byzantine fault tolerance (BFT) in both permissioned and permissionless blockchain systems. The key contributions of this chapter are the organization of these requirements and a new taxonomy that describes the requirements for security. The CAP Theorem is utilized to explain important tradeoffs between consistency and availability in the consensus algorithm design, which are crucial depending on the specific application of a given algorithm. This topic has also been explored previously by De Angelis [42].

4.1 Review of the Dissertation Statement and Research Questions

In this chapter, we provide the answer to the research questions below:

- **RQ-2**: What are the security requirements that enable a given consensus algorithm for blockchain systems to achieve liveness, safety, and byzantine fault-tolerance (BFT)?
- **RQ-3**: Do the preselected-leader consensus algorithms prioritize both consistency and availability according to the CAP Theorem tradeoffs?

Here, we provide a new taxonomy that describes the requirements for security (i.e., liveness, safety, and byzantine fault-tolerance (BFT)). And we utilize the CAP Theorem to explain important tradeoffs between consistency and availability in the preselected-leader consensus algorithms design.



Consensus correctness *
Liveness (Availability) **
Resistance to malicious miners DoS & DDoS attacks ‡
Termination ‡
Safety (Consistency) **
Validity ‡
Integrity ‡
Agreement ‡
Total order ‡
Crash tolerance
Unforgeable messages *
Confidentially **
Integrity **
Authentication **
Non-Repudiation **
The fraction of the network that can be compromised *
Synchronous & Partially synchronous t < n/2 **
Synchronous & Partially synchronous f < n/3 **
Crash tolerance
Byzantine tolerance
Performance *
Throughput **
Latency **
Consensus finality ‡
Scalability *
Participants' set size and dynamicity **
Performance **
Decentralization *
The barriers to entry to participate in the consensus **
Incentives *
Rewards for behaving well **
Penalties for behaving badly **
Resistance to Sybil attack *
The cost of creating multiple nodes **
Permissionless Blockchain
Public Permissioned/ Consortium Blockchain
Private Blockchain

Figure 4.1 The security-ingredients taxonomy for consensus algorithms in the blockchain.
* - Requirements, ** - Prerequisites to achieve requirements, ‡ - Further prerequisites,
`t` - The number of crashing nodes to be tolerated, `f` - The number of byzantine nodes,
`n` - The number of all nodes participating in the consensus (set size)

4.2 Considerations during consensus algorithm design

Some critical aspects to consider during the consensus algorithm design or evaluation are system fault resilience, each node's authorities (capabilities) with regard to ledger access, and the underlying network model for the distributed system hosting the particular blockchain. It is difficult to achieve fault resilience because of the faulty nature of nodes as well as the faulty nature of communication in a typical distributed system. Consensus algorithms must cope with node failures, network interruptions, malicious acts, and other unexpected behavior. Following are some of the fundamental aspects to consider in the design and evaluation to achieve a correct and byzantine fault-tolerant consensus algorithm for blockchain systems [33].

4.2.1 The network model

The level of coordination between the network nodes is defined by a given network's synchronicity. Network synchronicity determines the steps that the nodes need to take to reach an agreement and determines the level of system availability based on each node's behavior [33]. The second dimension of concern is termination, which means whether or not the nodes eventually decide on a value or not.

The network model levels are *synchronous*, *asynchronous*, and *partially synchronous*. In the synchronous network model, messages are assumed to be delivered within a fixed time; that property guarantees the termination of consensus, but it is not fault free. In the asynchronous model, messages may be arbitrarily delayed by an uncertain amount of time. Consensus algorithms in the asynchronous model are more resilient to faults than in the synchronous model, but termination is comparatively more of a concern [9, 112]. It is impossible to determine if a given node has failed or it is just taking a long time to respond; this result is stated in the "FLP impossibility result" by Fischer et al. [26]. Dwork et al. [45] proposed the partially synchronous network model, an asynchronous network model that eventually behaves synchronously (messages are delivered in an unknown but finite amount of time). Consensus protocol developers evidently prefer the partially synchronous (aka eventually synchronous) model because it is more realistic and since it is able to cope with probabilistic network behavior [33].

4.2.2 Type of failures

Failures in distributed systems may be either *crash* or *byzantine* failures. Consensus algorithms that tolerate failures are consequently classified as either crash tolerance or byzantine fault tolerance. The crash-failures set comprises a subset of the byzantine failures set. A crash tolerant consensus [40] algorithm reaches agreement even if one of the nodes experiences halting or is not responding, and despite communication collision(s), buffer overflow(s), and/or any other issue(s) during data transmission (e.g., Paxos and RAFT [73, 89]). A Byzantine fault tolerant consensus [40, 75] algorithm continues to operate despite the presence of faulty nodes, such as crash failures, and/or malicious nodes that cause conflicting information between nodes (e.g., PBFT [60, 75]). Any failure that hinders nodes from reaching an agreement on a value in a system is considered a byzantine failure.

4.2.3 Nodes membership and their access authorities to data

Other aspects to consider in the design of a consensus algorithm are node membership requirements for a given blockchain and their respective access authorities to the data in the blockchain. A blockchain system is classified based on the authority given to nodes and their participation in forming the consensus on appended blocks. Classification is also based on the type of node membership (i.e., public to everyone or private and limited to a subset of trusted nodes). Types of access to the data in a blockchain include *read, write, and commit*. Organizations design their particular blockchains according to the node membership type and data access authority [14]. A blockchain is *permissionless* if anyone is allowed to *read, write, and commit*, where nodes are neither trusted nor known, and membership is not required. Bitcoin is an example of a permissionless blockchain [83]. A blockchain is *permissioned* if a membership step is required for the node to join. Permissioned blockchains fall in one of three categories according to the node data-access rights. A permissioned blockchain is *public* if any node in the network has the right to *read*, but the right to *write* is only for the trusted nodes (members), and *commit* is limited to a subset of trusted nodes (members) such as in the Hyperledger fabric [31]. A *consortium* blockchain limits the right to *read* in the public permissioned blockchain to the trusted nodes (members). Corda [29] is an example of a consortium blockchain. A *private* blockchain limits the right to *read* to the trusted nodes (members) while the right to *write* and *commit* is controlled by the network administrator only. The authorities given to the nodes have a direct impact on the design of consensus algorithms, where the vulnerabilities increase with the size and dynamism of the set of nodes that are reaching consensus [42].

4.3 Consensus Algorithms Security Requirements and Prerequisites

Our thesis is that a secure consensus algorithm must achieve three criteria [33, 70, 71, 75][1]: liveness, safety, and byzantine fault tolerance. This section presents security requirements and their prerequisites that must be addressed in consensus algorithm design and evaluation while considering the assumptions in section 4.2. For example, the synchronicity of the underlying network has a direct impact on the right order of the submitted transactions and also how the agreement among the nodes is reached. Node membership and authority to access the data are two features that prioritize what must be considered in the design and evaluation of the consensus algorithms for both permissionless and permissioned blockchains, as well as their specializations. The organization of these requirements and a taxonomical figure that describes the requirements for security are shown in Figure 4.1.

4.3.1 Consensus Correctness

Consensus correctness is defined as the liveness and safety of the consensus algorithm. Lamport defined *liveness* as the good things that the algorithm should allow to happen, and *safety* as the bad things that the algorithm should not allow to happen to the system. In addition to that, a consensus algorithm is more reliable if it guarantees resilience to byzantine faults which include both crash and byzantine fault tolerance.

4.3.1.1 Liveness and safety of a crash-tolerant consensus

Liveness and safety properties are difficult to fulfill simultaneously. For any distributed system to be correct, it must reach some form of consensus on the correct answer. The three requirements—as in the Lamport scheme [72, 75]—for any correct consensus algorithm are validity, agreement, and termination. In the blockchain consensus algorithms seen in Figure 4.1,

[1]Lamport et al. [70, 71] defined safety and liveness simultaneously; Lamport et al. added [75] byzantine fault tolerance as well. Cachin [33] surveyed 13 consensus algorithms based on safety and liveness.

the safety requirements are validity, agreement, integrity, and total order; the liveness requirement is termination. Submitted transactions into the blockchain system must follow the network's deterministic and uniform rules (e.g., the transactions must follow the underlying network model's semantics such as synchronicity).

Validity means that a valid transaction is added into the blockchain if this transaction is received by all the honest nodes. In other words, the consensus algorithm cannot arbitrarily agree on some result or be hard-coded to always return the same value. *Agreement* means that majority of honest nodes should accept the submitted transaction and add it to the blockchain or else discard it. Honest nodes should maintain the same sequence number of blocks. In order for a given result to be meaningful, all functioning nodes must agree on it. It is pointless to consider the results of faulty nodes. *Integrity* means consistency of all accepted transactions in all honest nodes (which means no double spending[2]). All accepted blocks should be correctly generated and hash-chained in chronological order. Agreement and integrity together assure the *total order*, which is the sequence of blocks and transactions. *Termination*, the final requirement, means that all non-faulty nodes eventually will decide on the value. Termination is a liveness property that specifies what must happen for a system to be considered correct. Without the guarantee of termination, there is no guarantee that consensus will ever be achieved [72, 75].

The aforementioned requirements for consensus correctness are required with both fail-stop behavior and malicious behavior of the processes in the system in all blockchain types.

4.3.1.2 Liveness and safety of a byzantine fault-tolerant consensus

A byzantine fault-tolerant algorithm achieves liveness and safety in its design by achieving both properties that are provided by the CAP theorem [27] as well as those required in the crash-fault-tolerant design. The CAP theorem states that *"A system can only have two out of the three properties at any given time. A choice must be made between consistency, availability, and partition tolerance"* [27]. In proving the CAP theorem, there are situations where choosing two properties will force the system to yield on the third. The CAP theorem is not absolute; rather, it considers compromises. Almost every reasonable system will assume that partitions are

[2]Double spending is an attack where the attacker can be duplicated or has falsified certain digital information; it is a serious security threat in blockchains used to support cryptocurrencies [115].

going to occur in the system, which means that partitioning is a given. Consensus algorithms for blockchains, at a minimum, must be crash fault-tolerant. For this reason, the CAP theorem often reduces to a dilemma between consistency and availability. The choice is whether it is preferable for the system to return a false or outdated value or simply not to return a value at all. In a blockchain, the system is either resistant to forking the chain to different copies or resistant to denial of service attacks; this means it is either consistent or available, respectively.

Consistency means all nodes in the system maintain the same copy of the blockchain in the same order of blocks. This refers to the safety concept, which stops bad things from happening. If consistency is not achieved simultaneously, then it will be resolved later as in some consensus algorithms [2] where all nodes converge to the same copy. They converge to the longest chain as in Bitcoin and the GHOST protocol in Ethereum [110, 48]; this behavior is known as *eventual consistency*. Based on the CAP theorem, this type of blockchain is AP, which denotes the availability and eventual consistency type system. The *availability* of a blockchain is guaranteed if the submitted transactions are mined in blocks and eventually confirmed; thus the system achieves liveness.

In the preselected-leader consensus algorithms, miners agree to give the right of mining the block and add it to blockchain to one among them. Thus, the probability for blockchain forking is low as long as majority of miners are honest. Based on the CAP theorem, this type of blockchain is CP, which means it emphasizes strong consistency over availability. These consensus algorithms are vulnerable to some types of DoS attacks (e.g., insider Denial-of-service (DoS) attacks. They rely on one miner to mine and produce the block and that hinder the system availability because miners behave maliciously and decide to ignore the transactions or drop the blocks) [111].

4.3.2 Unforgeable Messages

The P2P network has four major concerns for the messages passing between the nodes; these are confidentiality, integrity, non-repudiation, and authentication. Consensus algorithms must consider these properties to maintain unforgeable messages [75]. *Confidentiality* in a blockchain means that both participants' identities and their transactions are protected against any malicious viewing, usage, or sharing. *Integrity* means that messages are not altered in transit and are

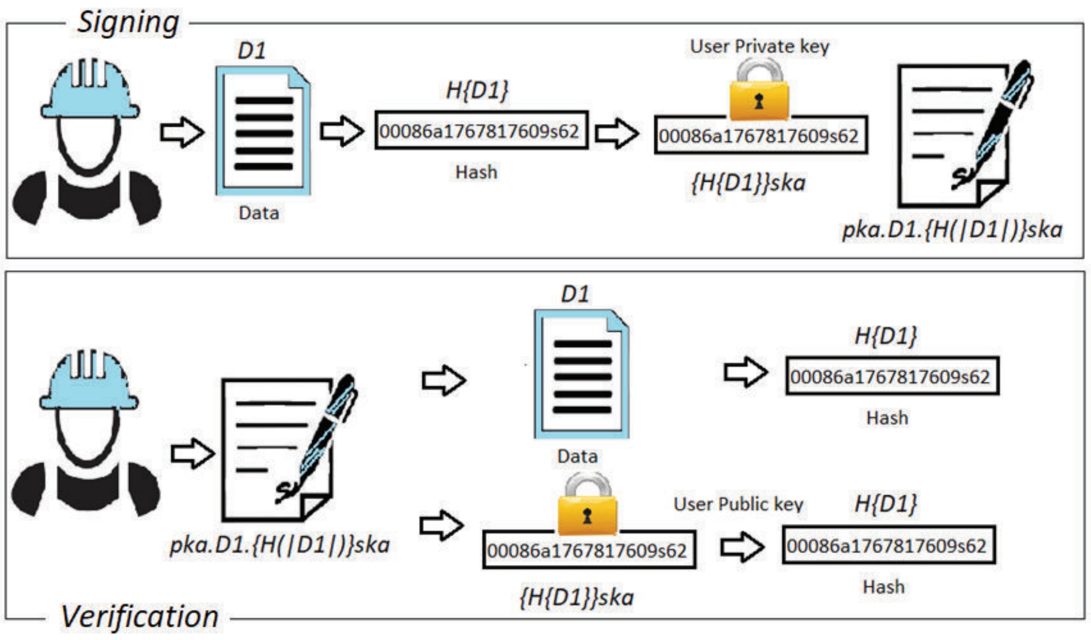


Figure 4.2 Digital Signature Diagram Adapted from [108]

complete. *Non-repudiation* means that the algorithm provides proof of the data origin (i.e., who created and mined the block). *Authentication* in blockchain means participants' identity is recognized without violating their privacy [7]. The aforementioned requirements are achieved by applying a cryptographic algorithm to verify and validate messages using, for instance, digital signatures [56].

4.3.2.1 Digital Signature

A system's integrity is guaranteed when any corruption or tampering of its data can always be detected [21]. Consensus algorithms use digital signatures to assure system integrity and ensure that messages are not altered in transit. These messages can include financial transactions, software distributions, or any other message where there is a need to detect data tampering or forgery. A digital signature is implemented using a combination of the Rivest-Shamir-Adleman (RSA) algorithm [64] and the SHA-256 cryptography hash function [43].

When a client sends a transaction $D1$ to a miner and the miner sends a block B1 (where B1 = Sign1, Sign2,..., SignN) to the blockchain, all parties obtain the evidence from the digital signature that the transaction was sent by the client and the block was sent by the miner, as illustrated in Figure 4.2. The miner can prove that the processed message was sent by the client by presenting $\{[D1]\}_{sk_a}$, which is the message signed by the client's private key [94]. Therefore, neither the sender nor the receiver of a message can deny transmitting or receiving the messages. Because digital signatures make deniability near impossible, they provide high confidence authentication.

Digital signatures are used in both preselected-leader consensus algorithms Lightweight Mining and BFT-Raft.

4.3.3 The Fraction of the Network that can be Compromised

Liveness and safety of the consensus algorithms depend on the failure models (crash and byzantine) and the common communications settings of the nodes (synchronous, asynchronous, and partially synchronous) [33]. Nodes in the network could be honest nodes that have failed or else be bad nodes that behave maliciously. Because of the difficulty of detecting malicious behavior, consensus algorithms tolerate different numbers of failed or bad nodes out of the total number of nodes while attempting to reach an agreement [18]. Consensus algorithms that are designed for asynchronous systems are more resilient to faults but they violate the consensus liveness criteria because of the lack of timely behavior and synchronization among the nodes. The "FLP impossibility result" states that it is impossible to guarantee consensus in an asynchronous network with the probability that one node might crash. Thus, the synchronous and partially synchronous network models are realistic environments and the termination property is guaranteed. Further, the number of nodes that the consensus algorithm can tolerate to guarantee the liveness and safety of the blockchain is *t crashed nodes* where $t < \frac{n}{2}$, *f byzantine nodes* where $f < \frac{n}{3}$ where n is the total number of nodes [72, 75].

4.3.4 Consensus Performance

Performance is measured by *throughput* and *latency* where throughput is the number of transactions processed in a given time while latency is the time it takes the transaction from the time it submitted to the time it confirmed [47]. Latency for users is how fast a transaction can be considered final, which refers to block confirmation. When an agreement on transactions is reached then the block is confirmed. Blocks that contain confirmed transactions cannot be deleted or changed. This is called *consensus finality*. Consensus finality relies on consistency; however, in cases where the algorithm follows eventual consistency, finality relies on the depth of the confirmed block and the cost to reverse the confirmed blocks [18]. For example, in Bitcoin, the default depth of the transactions to be confirmed is six blocks added to the chain including the current block. This is approximately less than or equal to 10% of the hash rate [12] required to avoid a double-spending attack. The complexity of the consensus algorithm has a direct impact on both throughput and latency. A secure design of a consensus algorithm must be byzantine fault tolerant and requires a minimum number of communication rounds to be achieved. This results in high transaction latency and low throughput. A permissionless blockchain (where the consensus algorithm is designed for a large number of nodes reaching an agreement with byzantine fault tolerance) is expected to have high latency and low throughput [33, 67]. Consensus algorithm designers must consider their system requirements carefully because optimizing throughput and latency in a byzantine-fault-tolerant consensus algorithm is challenging.

4.3.5 Consensus Scalability

Consensus algorithm scalability is measured by the impact on algorithmic performance with an increase in the *number of non-faulty nodes participating in the consensus* as well as the degree of the *dynamism resulting from increasing or decreasing the number of participant nodes in the consensus*. The flexibility for the participants to leave or to participate in the consensus requires more communication by the consensus protocol to determine the participants' set at a given time. The participants' set size and dynamism directly impact performance and latency, thus impacting consensus liveness [68].

4.3.6 Decentralization

Decentralization is a primary requirement in the permissionless blockchain. Barriers to entry measure how decentralized the blockchain is. The consensus algorithm is fully decentralized when it has minimal to no barriers for the node to participate in the consensus on the blocks [68]. The more barriers to entry that are needed to participate, the lower the degree of decentralization is, which limits the byzantine behavior of the nodes. For example, a private blockchain is a distributed system but one that is not decentralized since the authority to add and commit to the chain is controlled by the network administrator (vs. in Bitcoin where anyone can participate in the consensus).

4.3.7 Incentives

Incentives to participate in the consensus and the mechanism(s) for earning rewards have a direct impact on consensus correctness. Considering the net gain of participation is essential for consensus algorithm design. A Nash equilibrium [104] is the state of a system where participants' interactions are safe and benign for system stability, which means that a participant has no gain by changing their strategy while other participants' strategies remain fixed. Incentives could be designed by combining consensus goals with Nash equilibria and *rewarding good behavior* while *penalizing malicious behavior*. Economic incentives do not feature in classical consensus algorithms. Bitcoin, which uses proof of work (PoW) [83] consensus, introduced a financial incentive scheme for the miners to validate and mine blocks: Miners are rewarded with Bitcoins for their accepted work [67]. Positive and negative incentives have a direct impact on consensus correctness. Penalizing malicious behavior (i.e., negative incentives) affect blockchain safety. Rewarding good behavior (i.e., positive incentives) affect consensus liveness. Incentives have a bigger role in permissionless blockchains than in public blockchains. Consensus algorithms that are designed for permissionless and public blockchain need to have incentive schemes to provide system stability with a large number of participants. Thus, incentive schemes must also be considered when changing the degree of decentralization [113].

4.3.8 Resistance to Sybil attacks

A Sybil attack occurs when one person creates multiple identities and participates in the consensus as different independent nodes. Sybil attacks are only applicable for the permissionless blockchain in which nodes are unknown and untrusted. A Sybil attack has a harmful impact on consensus output [38]. If the attacker can create enough identities, then they can act maliciously and refuse to receive transactions or to mine blocks. In public permissionless blockchains, anyone possessing the requirements for mining can participate in the consensus since there are no constraints on node membership. Attackers can control the consensus result for their benefit; for instance, they can accept, drop, or even reverse transactions that lead to a double-spending attack [38]. It is important to consider *the cost that is needed for an attacker to create multiple fake nodes* and node memberships in consensus algorithm design. For example, in PoW, an attacker must have the majority of the hash rate in the network (more than 50%), which requires a large number of mining rigs (collections of powerful nodes) to accomplish that end. A Sybil-attack-resistant consensus algorithm is an algorithm that has a high cost of creating multiple accounts.

4.4 Security Requirements Based on Blockchain Nodes Membership

There are a number of security requirements that must be considered in any algorithm design and evaluation processes. The optimal secure design of a basic byzantine consensus algorithm is that which has the correctness (safety and liveness) and the highest byzantine fault tolerance requirements [75].

A private permissioned blockchain design needs a minimum level of security requirements for the consensus algorithm as described in Figure 4.1. In this type of blockchain, the authority to access, create, and/or confirm blocks is controlled by the network administrator. The miners' identities are known to the network administrator so the blockchain is not vulnerable to Sybil attacks. A private blockchain[3] is not decentralized and the consensus algorithm is designed for a fewer number of nodes; therefore, the confirmation of the transactions is faster because

[3]Some people argue that a private blockchain is not really a valid blockchain at all because it lacks any decentralization. We pass no such judgment here.

nodes are reaching consensus on the transaction faster. In this design, the consensus algorithm prioritizes consistency over availability when considering the presence of byzantine faults; hence, it is vulnerable to insider denial of service attack. The consensus algorithm that is designed for a private blockchain has high performance and low latency [91].

A public/consortium permissioned blockchain is semi-decentralized (access authority of blocks is granted to a predefined set of nodes instead of to a single authority). Permission to participate is still required; therefore, the number of nodes is not as large as the public permissionless network. The consensus algorithm designed for this type is more complex, which affects performance and scalability. Consensus algorithm complexity can be balanced using incentives to relax consensus algorithm complexity [115]. Thus, decentralization and incentives must be considered in addition to the security requirements that are considered for the private permissioned blockchain, as shown in Figure 4.1.

The consensus algorithm for public permissionless blockchain is designed for a large number of nodes and the challenge is to balance decentralization and byzantine fault tolerance. Anyone can join the network and participate in the consensus and the algorithm is vulnerable to Sybil attacks. Therefore, designers need to make it costly to create fake multiple nodes; for example, in the Proof of Work (PoW) [83] consensus algorithm, an attacker needs to have many expensive mining rigs to maintain the majority of the hash rate in the network. The consensus algorithm design grants availability over consistency when considering the presence of byzantine faults; hence, it is vulnerable to forking [16]. Thus, the consensus algorithm design follows eventual consistency and designers need to consider *consensus finality*, which relies on the depth of the confirmed block and the cost to reverse the confirmed blocks. Overall, tradeoffs of consistency vs. availability, and decentralization vs. performance and scalability must be considered when designing a secure consensus algorithm in addition to the application priorities.

4.4.1 CAP Theorem Dilemma in Blockchain

CAP Theorem is utilized to explain important tradeoffs in consensus algorithms design between liveness and safety, which are respectively consistency and availability. Where liveness (availability) is guaranteed if the algorithm is able to terminate and if the algorithm is resilient to the DoS attacks. And safety (consistency) is measured by the algorithm's ability to reach an agreement on the current state, maintain the total order of the transactions, and the algorithm resilience to forking [115].

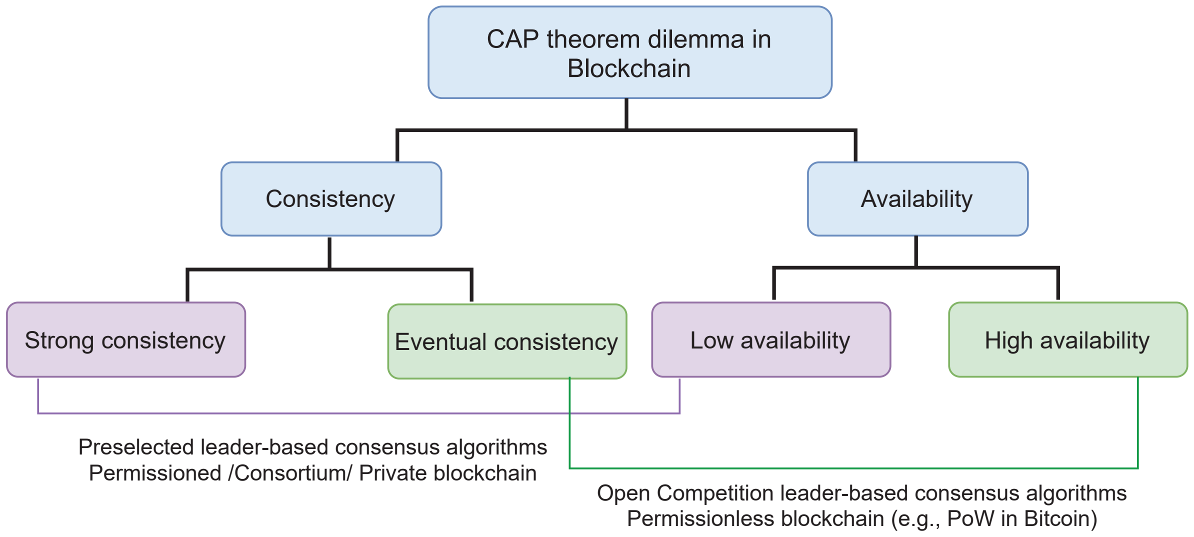

CAP theorem dilemma in Blockchain
Consistency
Availability
Strong consistency
Eventual consistency
Low availability
High availability
Preselected leader-based consensus algorithms
Permissioned /Consortium/ Private blockchain
Open Competition leader-based consensus algorithms
Permissionless blockchain (e.g., PoW in Bitcoin)

Figure 4.3 The CAP Theorem Dilemma in Leader-based Consensus Algorithms for Blockchain

In the blockchain space, system designers can use the CAP theorem to have a better understanding of the system and decide which of the two properties (availability or consistency) to prioritize since partitioning in the blockchain network is highly expected due to the nature of the network. The minimum security requirements of safety and liveness that must be guaranteed in consensus algorithms to achieve its goal are illustrated in Table 4.1. In this dissertation, we focus on the permissioned blockchain consensus algorithms where miners need permission to join the network and participate in the mining process. The preselected-leader consensus algorithms with no barrier are one example of the consensus algorithms that fit with this type of blockchain. In preselected-leader consensus algorithms, one leader miner is responsible for producing the block

Table 4.1 Security Requirements that Must be Guaranteed in any Consensus Algorithm

Consensus Correctness	CAP Theorem	Properties
Liveness	Availability	Resilient to DoS/DDoS Attacks and Termination
Safety	Consistency	Agreement, Integrity, and Total order

and adding it to the blockchain, thus they are considered strong consistency and low availability consensuses (i.e., the leader is the bottleneck of the system). The availability is low in these algorithms because both are vulnerable to DDoS/DoS attacks, see Figure 4.3. We deduce that the leader-based consensus algorithms are prioritizing consistency over availability and the algorithm developers need to address the liveness concern by adding steps to ensure the miners-selection process is totally random and fair to improve availability. The lightweight mining algorithm is a good example of taking steps to improve availability.

In this dissertation, we focus on preselected-leader consensus algorithms and their ability to maintain availability against malicious miners.

4.5 Summary

This chapter studied the "security ingredients" that enable a given consensus algorithm to achieve safety, liveness, and byzantine fault tolerance (BFT) in both permissioned and permissionless blockchain systems. The organization of these requirements and a new taxonomy that describes the requirements for security have been studied. We illustrated the use of Digital signatures to achieve message integrity and non-repudiation in blockchain systems. The CAP

Theorem was utilized to explain important tradeoffs between consistency and availability in consensus algorithm design, which are crucial tradeoffs depending on the specific application of a given algorithm. These are contributions of this dissertation, and also provide the answer to the research questions **RQ-2** and **RQ-3**.

CHAPTER 5

METHODOLOGY

This chapter describes the methodology design that we have developed to evaluate the availability of preselected-leader consensus algorithms against insider threats by malicious miners such as denial of services attacks. The evaluation process of using formal methods and established techniques for liveness (availability) will be implemented on two algorithms; the LightWeight Mining (LWM) and the byzantine fault-tolerant Raft (Tangaroa).

5.1 Review of the Dissertation Statement and Research Questions

The approach pursued here is designed to address the research questions described in Chapter 1. In the prior two chapters, we have addressed these research questions:

- **RQ-1**: Which consensus algorithms classification(s) (if any exist) is/are better for security evaluation (liveness and safety)?
- **RQ-2**: What are the security requirements that enable a given consensus algorithm for blockchain systems to achieve liveness, safety, and byzantine fault-tolerance (BFT)?
- **RQ-3**: Do the preselected-leader consensus algorithms prioritize both consistency and availability according to the CAP Theorem tradeoffs?

In this chapter, we build on the results given in Chapter 3 and 4, and address the remaining two research questions:

- **RQ-4**: What formal methods are used to test the liveness (availability) of preselected-leader consensus algorithms (LWM and BFT Raft) for a permissioned blockchain? Which work and which do not?

- **RQ-5**: What is the impact of the number of malicious miners on the liveness (availability) of the blockchain system? How is the system resilient to the malicious miner-based Denial of Service (DoS) attack?

Overall, our research statement, as stated earlier, is as follows:

1. Provide a common consensus algorithm classification based on how state changes are decided, focusing on the miner-selection process.

2. Identify “security ingredients” and map them to determine the ingredients that enable a given consensus algorithm to achieve liveness, safety, and byzantine fault-tolerance (BFT).

3. Using the Brewer’s Theorem (The CAP theorem), analyze the tradeoffs between liveness and safety in terms of availability and consistency, respectively, in consensus algorithms design.

4. Evaluate the preselected-leader algorithms (LWM and BFT Raft) for liveness (availability) against malicious miner DoS attacks using established techniques and formal methods such as Markov chain and queueing theory.

Note that Statements #1, #2, and #3 above were fulfilled by virtue of addressing RQ-1–RQ-3, as noted above. Addressing **RQ-4** and **RQ-5** will enable us to speak to Statement #4. In this chapter, we address the first part of the research question **RQ-4** on what the formal methods and techniques are used to evaluate consensus algorithms availability and the next part will be addressed in the next chapter. Also, using these methods in the analysis will assist in answering the research question **RQ-5**.

5.2 Methodology Design and Implementation

We provided the DLTs consensus algorithms classification based on how state-changes are decided in Chapter 3 to understand the miners-selection process. Understanding this process is the key to know how malicious miners could put the system’s availability and safety at risk. We studied the miner-selection process in two consensus algorithms, LWM and BFT-Raft, and classified them as preselected-leader consensus algorithms. In this type of algorithms, miners agree to abide by a fair-chance selection process where the right to mine the block is given to the selected miner (leader). However, malicious miners aim to disrupt this process illegally to gain unfair advantage and hinder other miners from reaching an agreement on who is the next miner.

In Chapter 4, we studied the security properties to achieve the liveness of a consensus algorithm, which are the resilience of consensus algorithms against malicious miners (availability) and termination. We utilized the CAP theorem to understand what the preselected-leader consensus algorithms prioritized between consistency and liveness, and we anticipated that leader-based consensus algorithms are vulnerable to DDoS/DoS attacks, thus they prioritize consistency over availability. However, the boundary between them is fuzzy. That is to say, various consensus algorithms have different mixes of availability and consistency, which together yield a near optimal level of security for the given system.

The miner's behavior during the mining process has a direct impact on the system's security. Malicious miners may manipulate and interrupt messages in several ways while good miners are trying to reach consensus or mining the transactions. Rayn [94] described the intruder's (malicious miner in our case) full capabilities of controlling the algorithms. There are number of possible behaviors that sabotage the liveness of the mining process, we present some of these behaviors that affect the consensus algorithm availability during the mining process below:

- Kill or delay messages: This is a type of denial of service threats, where malicious miners prevent the messages from being delivered or drop the messages at their end.
- Fake or replay messages: all messages must be signed using digital signature, also no one can deny sending the message. And any change or alteration to messages is detected since it is signed.

In this chapter, we study the availability of the two preselected-leader consensus algorithms against malicious miners. We illustrate the consensus algorithm's ability to tolerate malicious miners through determining the probability of good miners have the right to mine the blocks, and we calculate the average waiting time for the transactions submitted to the system to be mined within a reasonably accepted time. Queueing theory, Markov chain, and design a simulation are used to calculate the average waiting time for the transactions to be served in the presence of Byzantine nodes.

5.2.1 Threat Model

The threat model drives the security requirements of the consensus algorithms for blockchain systems. The proposed threat model covers a subset of attacks targeting the liveness and safety[1] of the consensus algorithms.

The proposed threats, as explained in the design methodology above, the formal methods, and the established techniques that we are being used to address in our methodology are shown in Table 5.1. Availability in this context is the ability of the consensus algorithm to tolerate malicious miners. And, digital signature is used to achieve messages integrity and non-repudiation, this is illustrated in Chapter 4.

Table 5.1 Threat Model

Security Requirements	Threats	Techniques
Availability	Malicious miner DoS attacks	Queueing theory and Markov chains
Integrity & Non-repudiation	Forging messages	Digital signatures

[1]Here, safety is covered in detail to the extent necessary for completeness; however, it's not a central part of this dissertation.

5.3 LWM Miner-selection Process

In LWM, miners agree to select a miner who becomes responsible for mining the block and broadcasting it to all other miners in the network. The miner-selection process as you see in Figure 5.1 starts where each miner generates and broadcasts the hash of a random value. Then, each miner waits for one-third of the hashes, and then the miner broadcasts the received set of hashes to all miners. Once values are received from the majority of the miners, then conflicting values are dropped and the miner(s) who sent them are dropped from the current round. This procedure is repeated when the miner shares a secret value, and then verifies that the secret value corresponds to the shared hash. And, finally, each miner calculates the summation of the received random numbers modulo the number of miners. Then a consensus is reached on the modulus result that is mapped to the id of one of the registered miners to be the selected miner or the leader.

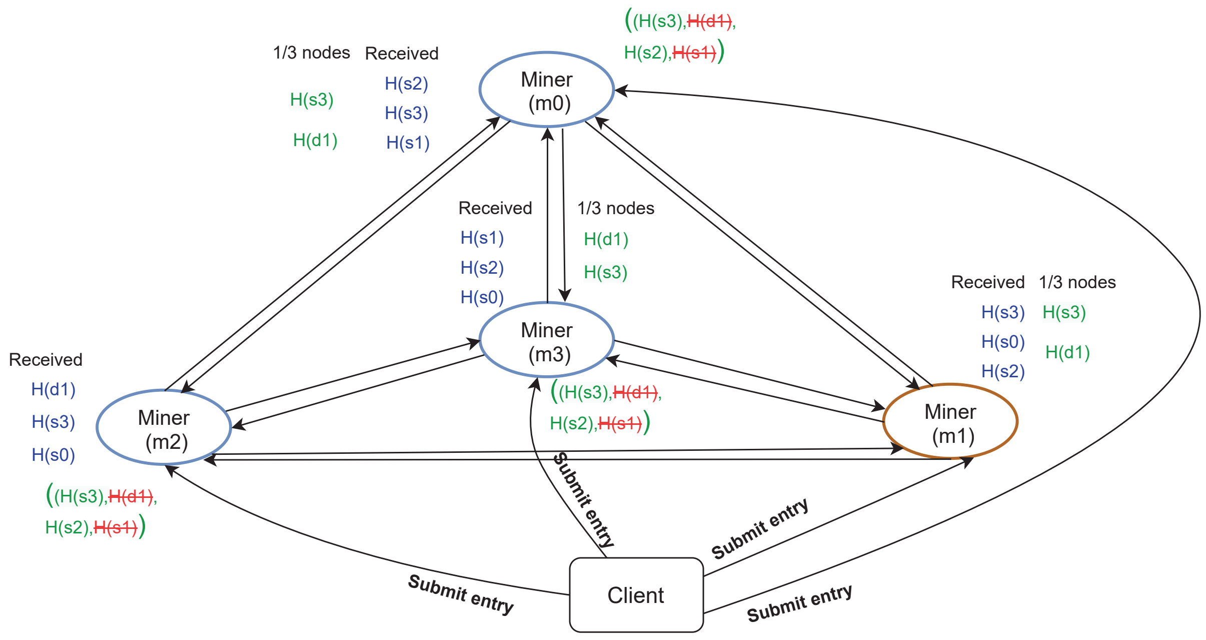


Figure 5.1 LWM Consensus Algorithm Miner-Selection Process

Malicious miners that act maliciously before being selected by sending conflicting values will be evicted from the round, however malicious miners might misbehave after being selected by ignoring and dropping transactions which affect the system availability. This is particularly significant because it is called out as an open problem in Narayanan et al.'s book [85].

The strength of the LWM algorithm is the ability to function correctly as long as there is at least one good miner (the $\sum_j r_j \mod N$ remains random). As long as the transaction is in one good miner's pool, the transaction will be mined, but it will take longer for the transaction to be added to the blockchain if there are a large number of bad miners. Therefore, to guarantee the miners are able to reach consensus, we assume the number of malicious miners is less than $\frac{1}{3}$ of the total number of miners. This assumption relies on Lamport's Byzantine Fault Tolerance (BFT) [75] results, which states that it is impossible to reach consensus if the number of faulty nodes, f is not less than $\frac{1}{3}$ of the total miners, n. This also assumes the percentage of miners participating in the mining process is greater than 50% of the total number of miners. The scope of this study includes the analysis of the system's availability against sabotage by malicious miners as a form of denial of services (DoS) attacks [85]. The scope excludes the analysis of the nodes' ability to converge to a decision despite the presence of arbitrary errors, and excludes the analysis of the ability of LWM to reach consensus despite the presence of malicious miners[2].

In LWM, when malicious miners drop valid transactions or erase transactions from their respective blocks, the system's availability is put at risk [85, 94]. However, this risk is mitigated because blockchains are distributed systems. Clients submit transactions to all the miners in the blockchain, so any honest miner can include the transactions when they create a new block.

5.3.1 Queueing Theory Analysis

The victim's transactions eventually arrive at the good miners' pool [57], which is proved using a Petri net and Reachability graphs[97]. In this dissertation we study LWM for the permission blockchain type, thus the algorithm is not vulnerable to Sybil attack. The probability of good miners being selected in LWM is an even chance and the leadership length term is limited to the block production time. Then, the probability of good miners being selected is given as below and

[2]This dissertation considers possible attacks in the system that are caused by miners delaying responses or/and forging messages. Timing issues and other aspects are covered by Bhat [20].

is denoted as $P_{G_{miners}}$:

$$P_{G_{miners}} = \frac{G_{miners}}{(G_{miners} + M_{miners})}. \tag{5.1}$$

In this dissertation, we are interested in the average waiting time before a client's transaction is added to the blockchain. To measure this, we utilized queueing theory. There are two types of clients: "regular clients," whose transactions are mined by any miner, and "victim clients," whose transactions are mined only by good miners. We assume that all transactions from regular clients and victim clients will end up in the miner's pool. When a good miner is selected, the transactions follow the first-come-first-serve (FCFS) rule. When a malicious miner is selected, victim client transactions are ignored, and only regular client transactions are served. Victim client transactions continue to wait in the good miners' pool until one of the good miners is selected to mine the next block. We assume in our analysis that the next miner-selection process will start after the current miner adds its block to the blockchain. A queue model is adopted to estimate the average waiting time for the victim client transaction to be added to the blockchain.

It is intuitive to conclude that when the number of malicious miners increases, the waiting time for victim clients' transactions will also increase. However, the expected waiting time is of interest. Scrybe is a single server model where only one miner can be active at a time. As described in Figure 5.2, we observe the average waiting time for the victim client transactions to be processed.

5.3.2 Theoretical Model

Scrybe follows a birth-death Markov process [79], which is a stochastic process that has a Markov property, also called a memoryless property. This means that one can only predict the future state based on the present state. A birth-death process is a continuous-time Markov chain [79] where the state variable increases or decreases by one, which represents the increase and decrease of the number of transactions. In our analysis we use the queueing theory model as a single service model, where only one miner can be active at one time. The transaction arrivals and departures follow the Markov process and the population is infinite. In the $M/M/1 : \infty/FCFS$ model, the $\mathbf{M}/M/1$ denotes the transactions that arrive with rate λ and follow the Poisson

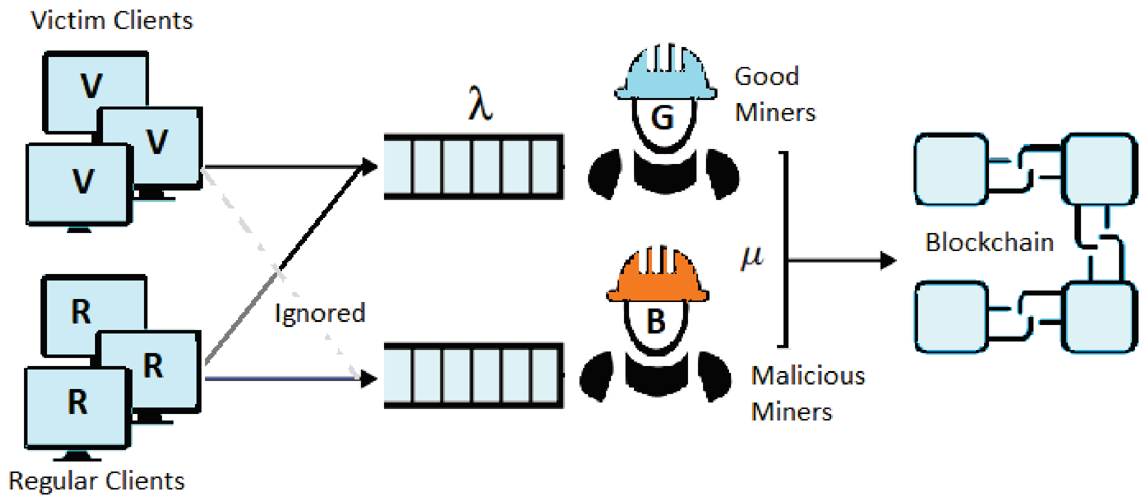


Figure 5.2 The Queue Model Logic

distribution [95], the $M/\mathbf{M}/1$: denotes the service times with rate μ which are assumed to follow the exponential distribution [95], and the $M/M/\mathbf{1}$ denotes one single service at a time. The capacity population of the system is assumed to be infinite and the queue protocol is first come first serve (FCFS). Since we are looking for the expected average waiting time for the victim transactions to wait until they are added to the blockchain, we use the system average waiting time formula as it is known in any basic queue model [95], given in **W** as follows:

$$\mathbf{W} = \frac{\lambda}{\mu(\mu - \lambda)} + \frac{1}{\mu}, \text{ where } \mu > \lambda. \tag{5.2}$$

In this analysis, we denote the number of regular clients as $R_{clients}$, the number of victim clients as $V_{clients}$, the number of good miners as G_{miners}, and the number of malicious miners as M_{miners}. The malicious miners ignore the victim client transactions in their pool and either include other client transactions or include nothing in the block that is being mined. This will cause a delay before the victim client transactions are served, so we are interested in finding the average waiting time for both regular and victim client transactions in the good miners' pool. We assume all clients submit their transactions at the same rate, so we set an arbitrary number for the client transaction generation rate as gen_{rate} which is 6,000 transactions per second. Additionally, we assume that all miners mine at the same rate, so we set the average mining rate as **r** equals five transactions per second. Using the above conditions, the average arrival rate $\lambda_{G_{miners}}$ for the good miners' pool is given in the following equation:

$$\lambda_{G_{miners}} = gen_{rate} \times \left(V_{clients} + R_{clients} \times \frac{G_{miners}}{(G_{miners} + M_{miners})} \right). \tag{5.3}$$

The average service rate $\mu_{G_{miners}}$ is the reciprocal of the expected average waiting time. We find $\mu_{G_{miners}}$ as follows:

- Each miner has the same chance of being selected, so the probability that the selected miner is good is given in $P_{G_{miners}}$ as follows:

$$P_{G_{miners}} = \frac{G_{miners}}{(G_{miners} + M_{miners})} \tag{5.4}$$

- Thus, the probability that a malicious miner is selected, denoted by P'_1, is the probability of the victim client transactions being delayed, which is as follows:

$$P'_1 = 1 - P_{G_{miners}} \tag{5.5}$$

- Accordingly, the probability of a victim transaction being delayed twice in a row is denoted by P'_2 as follows:

$$P'_2 = (1 - P_{G_{miners}})^2 \tag{5.6}$$

- Therefore, the probability of a victim transaction being delayed n times in a row is denoted by P'_n as follows:

$$P'_n = (1 - P_{G_{miners}})^n \tag{5.7}$$

Given that the average mining time for any transaction to be mined is $\frac{1}{r} = 0.2$ and using the expected value formula [93], which is the probability of a transaction to be delayed multiplied by the number of delays multiplied by the amount of time for each delay.

- The expected value when there are multiple probabilities of delays, where t is the number of delays, is given in equation 5.8, which simplifies to equation 5.12:

$$E_{(t)} = \frac{1}{r} \sum_{n=1}^{\infty} n \times (1 - P_{G_{miners}})^{n}, \text{ where } 0 < P_{G_{miners}} < 1 \tag{5.8}$$

Divided both sides by $(1 - P_{G_{miners}})$, will get:

$$\frac{E_{(t)}}{1 - P_{G_{miners}}} = \frac{1}{r} \sum_{n=1}^{\infty} n \times (1 - P_{G_{miners}})^{n-1}, \text{ where } 0 < P_{G_{miners}} < 1 \tag{5.9}$$

Using the variant [103] for the following geometric series:

$$\sum_{n=1}^{\infty} n \times x^{n-1} = \frac{1}{(1-x)^2}, \text{ for } |x| < 1 \tag{5.10}$$

And substituting into equation 5.9:

$$\frac{E_{(t)}}{1 - P_{G_{miners}}} = \frac{1}{r \times (-P_{G_{miners}})^2} \tag{5.11}$$

Thus,

$$E_{(t)} = \frac{1 - P_{G_{miners}}}{r \times (P_{G_{miners}})^2} \tag{5.12}$$

- The average time for a transaction to be served, denoted $\mu_{G_{miners}}$, is $\frac{1}{E_{(t)}}$ as in equation 5.13, where $\mu_{G_{miners}}$ is the mean service time elapsed before the victim client transaction is added into the blockchain.

$$\mu_{G_{miners}} = \frac{r \times (P_{G_{miners}})^2}{1 - P_{G_{miners}}} \tag{5.13}$$

Table 5.2 Simulation Parameters

Parameter	**Symbol**	**Values**
Transactions	T	1K , 30K
Malicious miners	M_{miners}	1 to 10
Good miners	G_{miners}	$\lvert 10 - M_{miners} \rvert$
Victim clients	$V_{clients}$	1 to 1000
Regular clients	$R_{clients}$	$\lvert 1000 - V_{clients} \rvert$
Iterations	i	1 to 45
Mining rate	r	5 BPS
Confidence interval percent	CI	95%
Transactions generation rate	gen_{rate}	6,000 TPS

5.3.3 Simulation Model

The previous theoretical approach is implemented as a discrete event simulation (DES) [35] in Python. It simulates the queue model using both calculated $\lambda_{G_{miners}}$ and $\mu_{G_{miners}}$ based on the logic above, and the simulation time is advanced based on the event that is closest in occurrence (see Figure 5.3). We ran 45 iterations of the simulation with 1,000 transactions, 10 miners, and 1,000 clients. The results converged after 45 runs of each scenario. We designed the simulation for several scenarios based on different numbers of malicious miners and different percentages of victim clients, simulations parameters which are used as inputs are shown in Table 5.2. We calculated the average waiting time by summing the difference between departure time and arrival time for all transactions then dividing by the number of transactions. We adopted the 95% confidence interval [77] to find the accuracy and the uncertainty of the estimated average waiting time for our samples. This is calculated for each scenario (1,000 transactions) and then averaged out for all 45 iterations. This is shown mathematically in equations 5.14, 5.15, 5.16, and 5.17, where i represents a scenario for 1,000 transactions that is repeated 45 times for a specific number of malicious miners and a specific percentage of victim client transactions:

$$\mathbf{X}_i = Mean_{1000(Transactions)}, \tag{5.14}$$

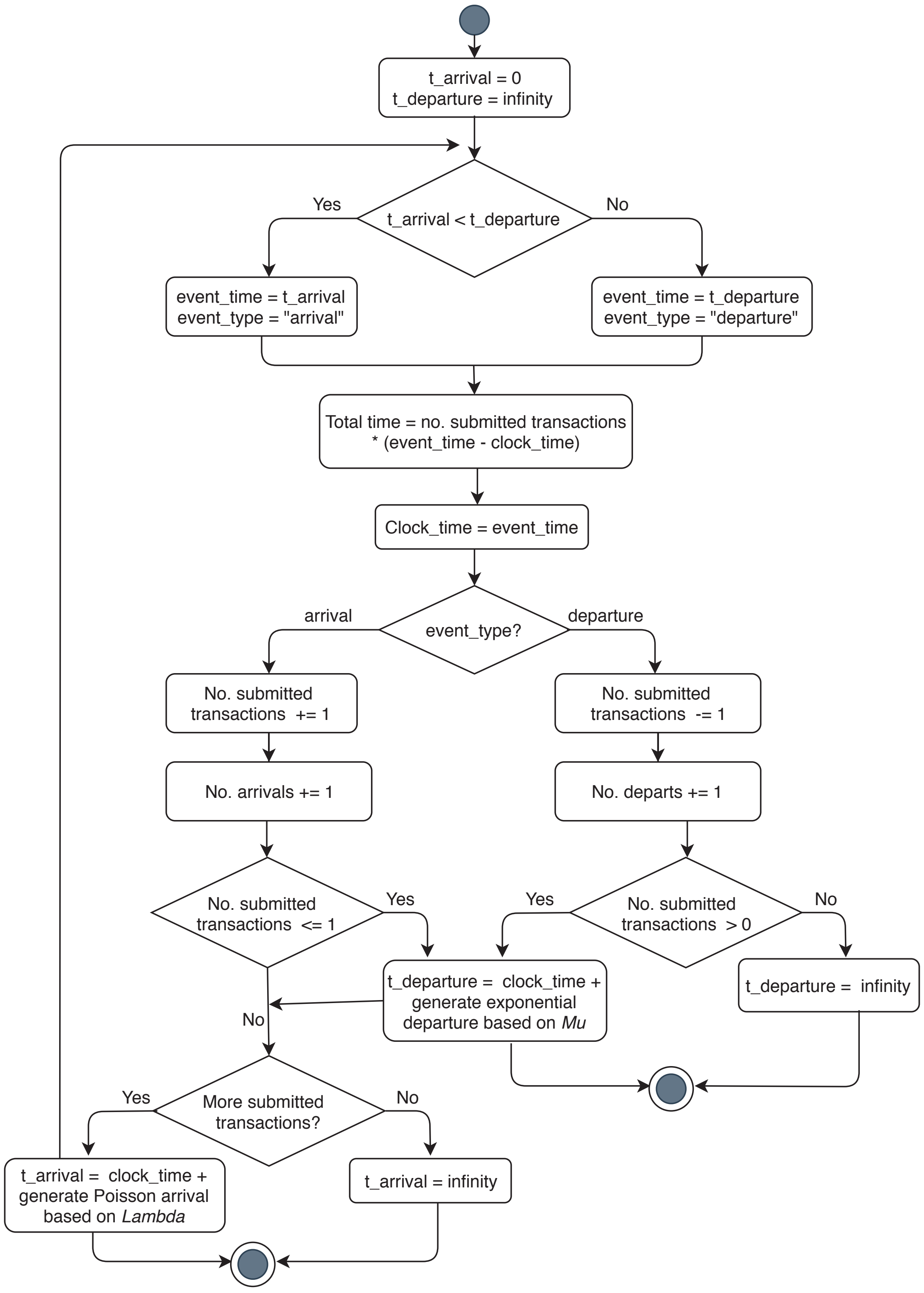

t_arrival = 0
t_departure = infinity
t_arrival < t_departure
Yes
No
event_time = t_arrival
event_type = "arrival"
event_time = t_departure
event_type = "departure"
Total time = no. submitted transactions * (event_time - clock_time)
Clock_time = event_time
event_type?
arrival
departure
No. submitted transactions += 1
No. submitted transactions -= 1
No. arrivals += 1
No. departs += 1
No. submitted transactions <= 1
Yes
No
No. submitted transactions > 0
Yes
No
t_departure = clock_time + generate exponential departure based on Mu
t_departure = infinity
More submitted transactions?
Yes
No
t_arrival = clock_time + generate Poisson arrival based on Lambda
t_arrival = infinity

Figure 5.3 Queue Model Simulation Algorithm Adapted from [44]

$$\mathbf{Mean}_i = \frac{\sum_{i=1}^{45}(\mathbf{X}_i)}{45}, \tag{5.15}$$

$$\sigma_i = \sqrt{\frac{\sum_{i=1}^{45}(\mathbf{X}_i - \mathbf{Mean}_i)^2}{45}}, \tag{5.16}$$

$$\mathbf{95\%\ CI}_i = \frac{1.96 \times \sigma_i}{3\sqrt{5}}. \tag{5.17}$$

5.4 BFT-Raft Miner-selection Process

Tangaroa (BFT Raft) is similar to Raft in the miner-selection process, and it's called leader election stage. In BFT-Raft, the leader is responsible for ordering the blocks (the leader is also responsible for the order of transactions in the block) in the other miners' copies. Leaders fail either due to system crash or by a client requesting reelection because of the leader's malicious behavior. The initial state of the system as you see in Figure 5.4 is when all miners are followers and then the leader-election stage starts. Each miner/follower picks a random timeout interval between 150 and 300. Once the time runs out, the follower nominate itself (by voting to itself) to be a leader, all the other nodes check if leader is elected or not, and if there is no elected leader, they vote for the nominee. The candidate with most votes become the leader. When a leader is chosen, log replication stage is triggered. In this stage, the leader accepts and processes the requests from clients.

Tangaroa (BFT Raft) does not have a verification process to verify the randomness of the time interval values, so it violates the liveness criteria if a malicious node rushes its timeout at time zero (i.e., at time 150) to become the leader. The malicious node can refuse to work or delay the service when it becomes a leader causing a violation of the liveness criteria.

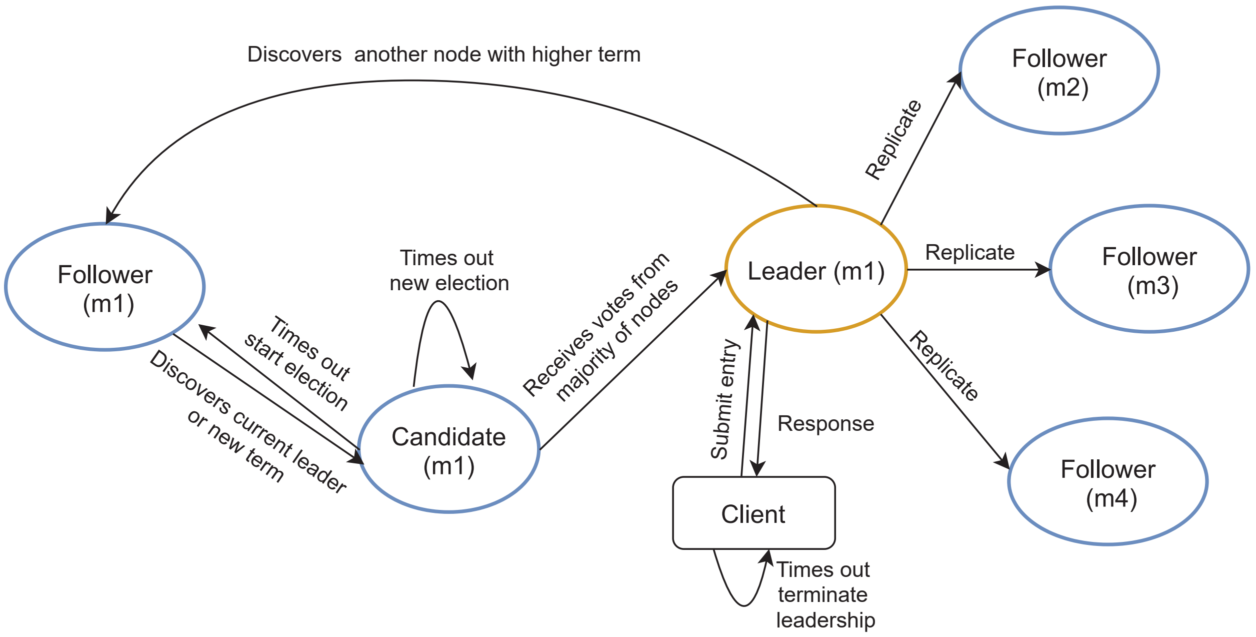


Figure 5.4 BFT-Raft Consensus Algorithm Miner-Selection Process

5.4.1 Theoretical Model

Each miner randomly picks its own time interval to wait in the range $[150, 300]$. A bad miner rushes at time zero. There is always a possibility that a good candidate will randomly get 150 at the same time as the bad candidate rushes its time. Ignoring the scenarios[3] where more than one good miner getting 150, we calculate the probability of one good candidate getting 150 (i.e., two miners (1 good, 1 bad) getting 150 and become candidates). This is the percentage of the time that two candidates (bad, good) are competing for leadership, this percentage is denoted as Y. This percentage is split fairly between the two candidates. Therefore, the probability of the good miner becoming the leader is $\frac{Y}{2}$. And the probability of the bad miner becoming the leader is $1 - \frac{Y}{2}$.

[3]This is for simplicity of illustration; however, the actual programming and equations include all scenario where (one, two, ..., # of good miners) are also getting to wait at 150 at the same time

To explain this logic theoretically, we use the binomial distribution to calculate the probability of a malicious miner winning the leadership. The binomial distribution formula is denoted as b(x), where n is the number of the total number of trials, p is the success rate, q is the failure rate, and x is the total number of successes:

$$b(x) = \sum_{x=1}^{n} \binom{n}{x} \times (p)^{x} \times (q)^{n-x} \tag{5.18}$$

Applying that to the BFT-Raft scenario; x is the number of good candidates who are randomly pick the value 150, G is the total number of good miners, the success rate p is equal $\frac{1}{150}$, and the failure rate q equals $1-p$. Below are the equations are used to determine the percentage of time a good miner wins, the variables used are as described above.

$$b(x) = \sum_{x=1}^{G} \binom{G}{x} \times \left(\frac{1}{150}\right)^{x} \times (1-p)^{G-x} \tag{5.19}$$

The probability that good miners getting the value 150 randomly (i.e., being candidates) equals b(1) + b(2) + ...+ b(G). So, the total probability of competition (at the same time with the bad candidate rushing its time) is given in $S_{(G)}$.

$$S_{(G)} = \sum_{x=1}^{G} b(x) \tag{5.20}$$

The probability of a bad miner wining when other candidates are present is fairly prorated and is denoted as r as shown below:

$$r = \sum_{x=1}^{G} \frac{b(x)}{x+B} \tag{5.21}$$

Thus, the probability of a bad miner winning which denoted as $W\prime_B$ equals:

$$W\prime_B = 1 - (S_{(G)} - r) \tag{5.22}$$

Considering bad miners colluding, the probability of a bad miner wining with other bad miners colluding (i.e., voting for the bad candidate) and when other candidates are present is prorated based on the number of colluding miners is denoted as W_B and shown as below:

$$W_B = \sum_{x=1}^{G} b(x) \times \frac{B}{x+B} \tag{5.23}$$

5.4.2 Simulation Model

The previous theoretical approach is implemented in Python. We simulate the BFT-Raft Leader election stage with the scenario of one malicious miner constantly picks the value 150. In cases of more than one malicious miner in the system and other good candidate also get the 150, malicious miners collude and vote for the malicious candidate to increase its chance of winning. We ran 1000 iterations of the simulation with 10 miners. We designed the simulation for several scenarios based on different numbers of malicious miners. We calculate the number of times that the malicious miner winning by getting the majority of votes.

5.5 Summary

In this chapter, we described our approach to answering our research questions and thereby demonstrating our dissertation statement. We illustrated the design and implementation of the evaluation methodology to test for the availability of the consensus algorithms. We implemented this methodology on two consensus algorithms: LightWeight Mining (LWM) and the byzantine fault-tolerant Raft (Tangaroa).

We explained the theoretical and simulation models used to analyze the ability to tolerate malicious miners for both. This chapter provided the answer for the research question **RQ-4** on what the formal methods and techniques are used while the next chapter will address the second part of this question and illustrate the result and provide other formal methods that did not work. The result of the analysis provided in this chapter will assist in obtain the answer to research question **RQ-5**, which is provided in the next chapter.

CHAPTER 6

ANALYSIS OF RESULTS

In this chapter, we provide the generated results and their analysis based on the implementations that were performed to evaluate consensus algorithms availability; in particular for LightWeight Mining (LWM) and BFT-Raft algorithms. We present the results generated by both the theoretical model and simulation models, and we analyze them based on their ability to tolerate malicious miners. We also explain the formal methods that failed to work for this evaluation.

6.1 Review of the Dissertation Statement and Research Questions

In the prior two chapters, we have addressed the research questions **RQ-4** and **RQ-5** partially, and in this chapter, we address them completely, these research questions are as follows:

- **RQ-4**: What formal methods are used to test the liveness (availability) of leader-based consensus algorithms (LWM and BFT Raft) for a permissioned blockchain? Which work and which do not?
- **RQ-5**: What is the impact of the number of malicious miners on the liveness (availability) of the blockchain system? How is the system resilient to the malicious miner-based Denial of Service (DoS) attack?

6.2 LWM Availability Analysis Results

The average waiting time calculated in the queue model simulation approximately matches the theoretical average waiting time, with 95% of the 45 means within the confidence interval. The average waiting time for the victim client transactions increased exponentially as the percentage of malicious miners increased, see Figure 6.1, and the sample data in Table 6.1. We applied several steps to verify the simulation results for each scenario; these steps are summarized as follows:

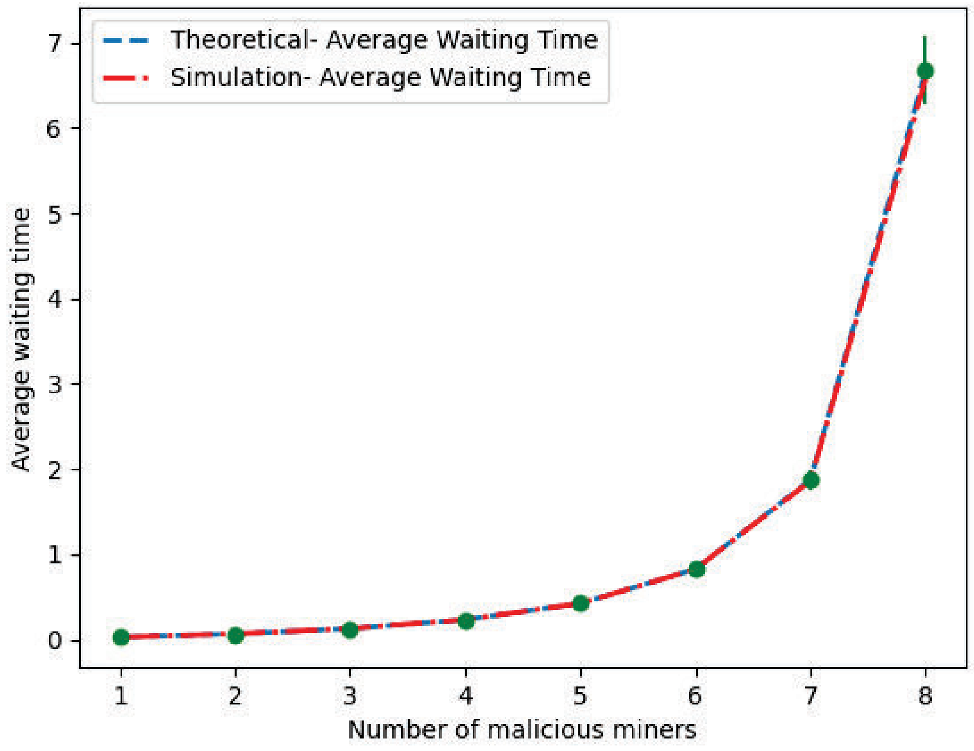


Figure 6.1 Average waiting Time vs. Number of Malicious Miners with 50% Victim Transactions

- Calculate the log_{10} of the average waiting time resulted by the simulation.
- Apply linear regression [100] to generate an equation that predicts the average waiting time with different number of malicious miners.

$$\mathbf{y}_i = b + ax_i, \text{ where the } x_i \text{ represent the number of malicious miners} \tag{6.1}$$

- Calculate values based on above developed formula.
- Calculate the anti-log which is $10^{\mathbf{y}_i}$, where $\mathbf{y}_i$ is the generated value from equation 6.1.

Table 6.1 Simulation Results Sample

Inputs		Generated by simulation			Outputs		
$V_{clients}$	M_{miners}	$P_{G_{miners}}$	$\lambda_{G_{miners}}$	$\mu_{G_{miners}}$	S_{Wait}	Th_{Wait}	95% CI$_{size}$
50%	1	0.9	0.158	40.500	0.025	0.025	0.002
	2	0.8	0.150	16.000	0.063	0.063	0.004
	3	0.7	0.141	8.167	0.125	0.125	0.008
	4	0.6	0.133	4.500	0.229	0.229	0.014
	5	0.5	0.124	2.500	0.420	0.421	0.026
	6	0.4	0.116	1.333	0.824	0.821	0.051
	7	0.3	0.107	0.643	1.854	1.867	0.115
	8	0.2	0.099	0.250	6.573	6.614	0.408
	9	0.1	0.090	0.056	3530.911	-28.768	126.611

Points generated from the equation resulting from the data linear regression are plotted in Figure 6.2 along with the simulation data. The graphs are done in both linear and log scales, we explained how we plotted the error bars in log scale in AppendixA . Both show a reasonable match between both sets of data up to eight malicious miners, at which point they begin to diverge. When there were nine malicious miners and the system and simulation broke down, the results did not match.

To show singularity, we ran the simulation for several cases with the scenario of nine malicious miners to find when the system breaks down. This allowed us to discover the maximum waiting time that the victim client should wait to be served, and what percentage of miners can be malicious before the system breaks down. Cases are as follows: running the simulation for 1,000 transactions, 30,000 transactions, and fixed λ to be 0.055556, which is close to μ. The analysis shows that the system breaks down when 20% - 30% of the clients are victims. Using linear interpolation, we found that the system singularity occurs when the system is 26% victim clients and has nine malicious miners.

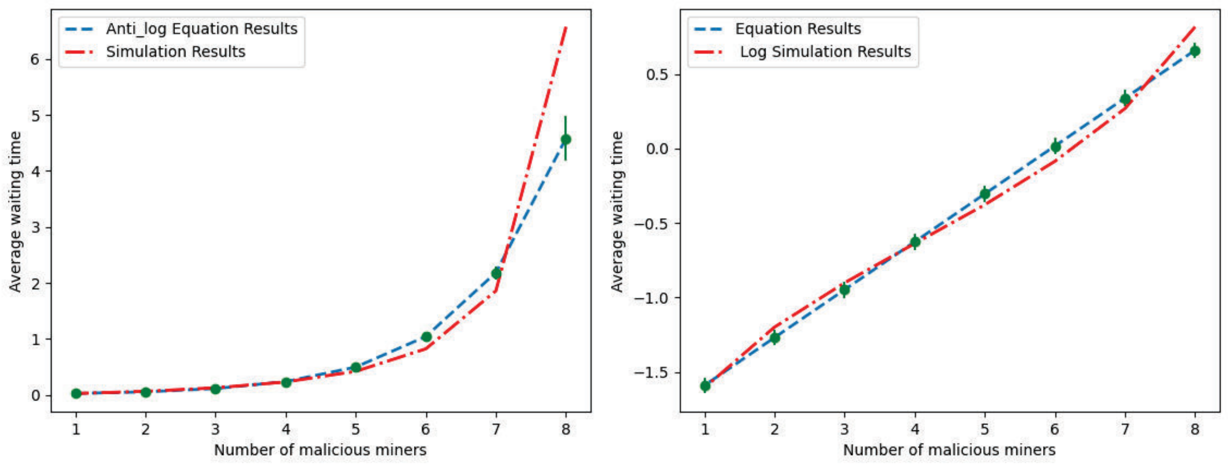


Figure 6.2 Average waiting Time vs. Number of Malicious Miners with 50% Victim Transactions, Linear Equation and Simulation Data

6.3 BFT-Raft Availability Analysis Results

In this section, we analyze the result for the simulation and the theoretical logic implemented in the prior chapter and we provide proof that BFT-Raft is vulnerable to malicious miners' DoS attacks. Below is the theorem and the proof of this claim.

Theorem. BFT-Raft with its current setup that does not protect against malicious miners rushing the time intervals is extremely vulnerable to DoS attacks, such that one malicious miner can halt the system.

Proof. The first step to prove this is to determine the probability of a malicious miner winning and become the leader when rushes its time interval. Because even though the malicious miner constantly rushes the time and picks the value 150, there is a chance exists that other good miners will also get the value 150 randomly. For this type of probability, the binomial distribution is used to determine the success rate for the malicious miners as well as the good miners. To do that, we worked indirectly with binomial distribution by calculating the probability of good miner(s) getting the value 150. When this happens, the malicious miner is sharing the success probability with other good miners, and for the rest of the time, the malicious miner does not have any competition. This is illustrated in Figure 6.3 below:

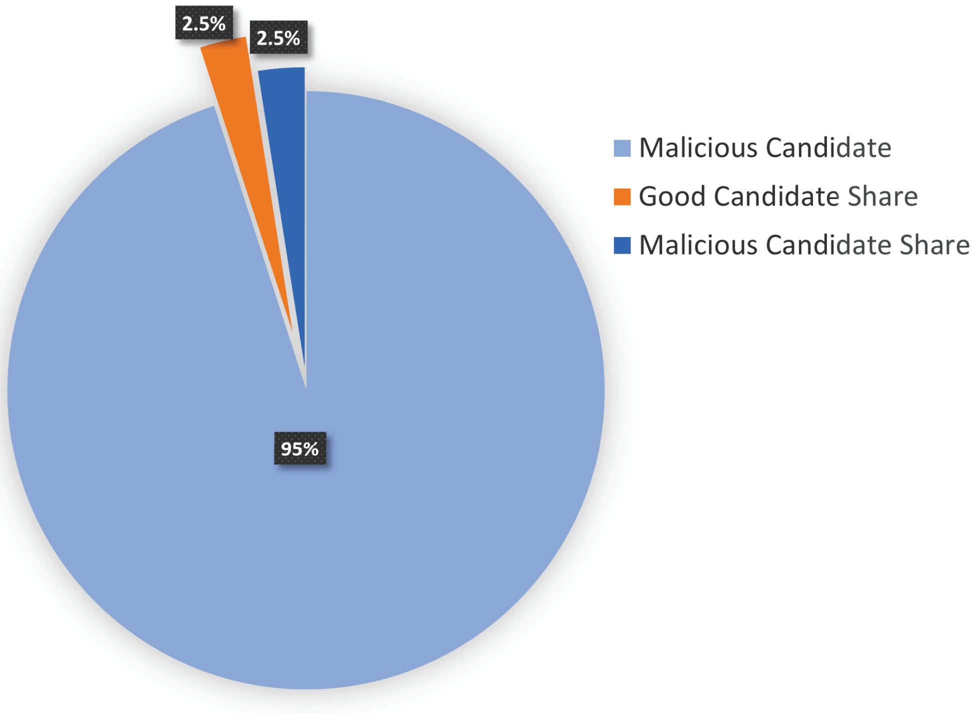


Figure 6.3 Probability of One Malicious Miner Winning in BFT-Raft

Using the above logic the probability of a malicious miner succeeding is calculated using equations 5.19 through 5.8, in BFT-Raft theoretical model section. As shown in Figure 6.4 generated by the simulation and the theoretical results, one malicious miner succeeds to be the leader about 95% of the time, which results in paralyzing and halting the system.

It gets even worse of other malicious miners who are colluding together and disrupt the system. To dig this a step further, we ran a simulation of the system to estimate the average waiting time for the transactions to be added to the blockchain as a result of malicious miners' presence, here we consider those who get the chance to be mined by the good miner.

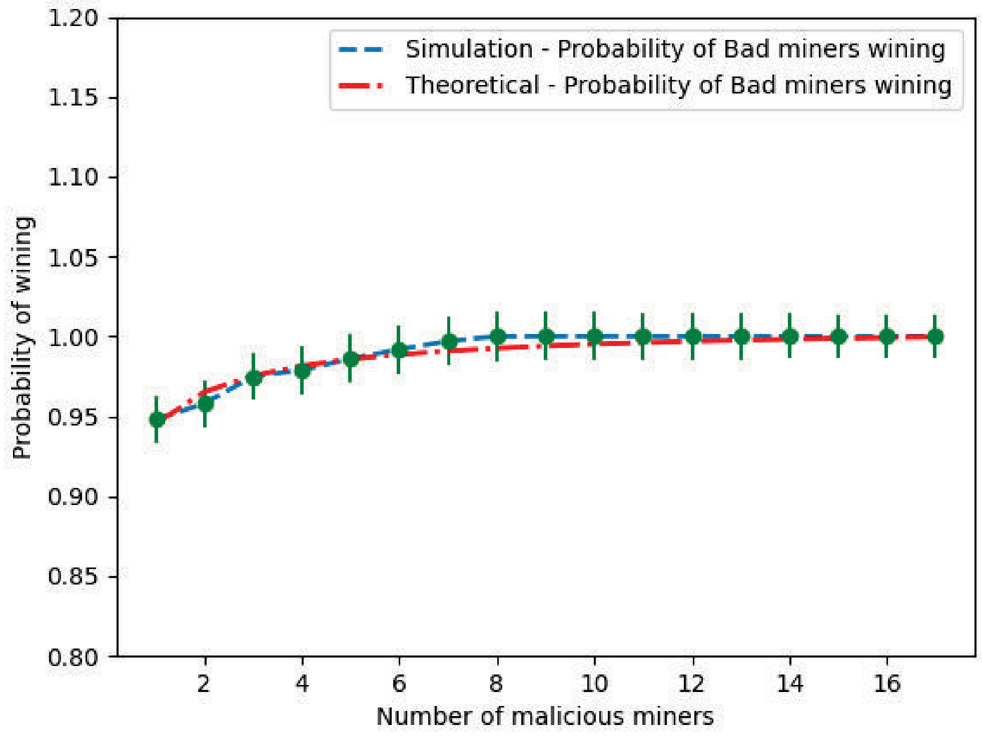


Figure 6.4 Simulation and Theoretical Results of One malicious Miner Winning in BFT-Raft

The simulation results showed that even with one malicious miner, the average waiting time is in hours and increases exponentially with the number of malicious miners. Because of these limits, practically no transactions reach the good miners for processing, which makes μ the departure rate to approach zero and be less than λ the arrival rate. Based on the fact that μ is less than λ 95% of the time, we find that the queueing theory does not work for BFT-Raft. without fixing the rushing time gab, BFT-Raft is not byzantine fault-tolerant.

6.4 Lightweight Mining vs. Proof of Elapsed Time vs. BFT-Raft

For completeness, in this section, we make a cursory comparison between LWM, PoET, and BFT-Raft. These three algorithms are preselected-leader consensus algorithms, where the miner selection process is designed as a random leader election or a lottery based election model. In this dissertation, we focus on consensus algorithms for permissioned blockchain, thus we provided the liveness analysis for LWM and BFT-Raft as consensus algorithms for this type of blockchains where the systems is not vulnerable for Sybil attack [33]. PoET is a consensus algorithm that is designed for permissionless blockchain, however it is often used for permissioned blockchain such as Hyperledger Sawtooth.

As mentioned earlier in Chapter 3 that nodes in PoET are each given a random amount of time they must wait before creating a new block. Therefore, the node that is assigned the smallest amount of time will be allowed to create the next block and broadcast it to all other nodes in the network. To ensure that all random numbers are assigned fairly, Intel requires that each node has specialized hardware called the Software Guard Extensions (SGX) [5]. Breaking this trusted specialized hardware (SGX) security and the manipulation of its fairness increases the malicious miner probability of winning and increases its chance of being the selected miner to mine the next block. Also, a malicious miner may collect more chips (SGX) as that increases its chance or probability to be selected, thus every additional chip is another chance of winning [80].

By intuition, we deduce that hacking the Software Guard Extensions (SGX) chip is harder than the BFT-Raft protocol for the miner who selects its time interval. Therefore, it is expected that PoET will perform much better than BFT-Raft especially that we have demonstrated that BFT-Raft practically is halted by a single malicious miner.

Both LWM and PoET offer a good level of randomness in selecting the miner, however, LWM is more robust in the sense that hardware is more susceptible to hacking. PoET uses a chip that is designed by Intel is expected to be safe, secure, and hard to crack, but the possibility to be hacked exists. This can be turned into a low probability high impact event. Taking the proceedings into considerations and taking into account the security of the Software Guard Extensions (SGX) chip is out of the scope of this dissertation, we infer that LWM is more robust than PoET. The result of this cursory review is that LWM is more robust than PoET, and PoET is more robust than BFT-Raft.

6.5 Formal Methods that Failed to Work

In this section, we present the formal methods that were used to analyze the consensus algorithms availability and failed to work, and here we provide our feedback on methods that failed to to test for finding the time it takes the transactions to be added to the blockchain. Our approach is implemented using Petri nets, Reachability graphs, and the Ford-Fulkerson algorithm (FFA) for LWM.

A Petri net is used to model, visualize, and help analyze behaviors having parallelism, concurrency, synchronization, and resource sharing [41]. A Petri net is a collection of directed arcs connecting places and transitions. Places that are represented by circles may hold an integer (positive or zero) number of tokens or marks. The marking of a net is its assignment of tokens to places. Transitions are represented by bars. Each arc is associated with a fixed weight. A transition is enabled when the number of tokens in each of its input places is at least equal to the arc weight going from the place to the transition. An enabled transition may fire at any time. When a transition is fired, the tokens in the input places are moved to output places, according to arc weights and place capacities. The input arc weights are subtracted from the input place markings, and output arc weights are added to the output place markings. A technique to analyze these nets is to generate all possible states in one graph called the Reachability graph. Figure 6.5 is the Petri net model designed for LWM considering the scenarios with the presence of more than one malicious miner.

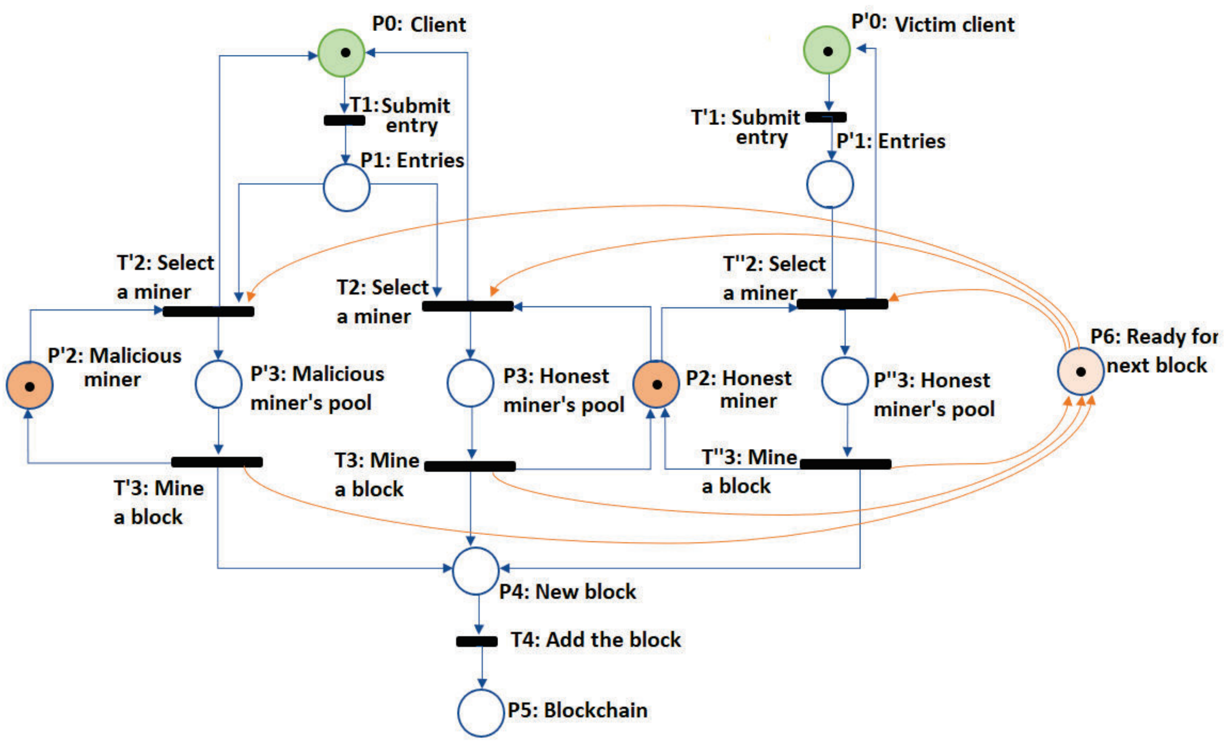


Figure 6.5 Petri Net for the LWM Algorithm with the Scenario of Multiple Malicious Miners

We applied the Ford-Fulkerson Algorithm (FFA) on the generated reachability graph[1] to find the maximum capacity for the augmented path[2] from the source (start node), which is the victim client, to the sink (end node), which is the blockchain itself. FFA-based analysis did not result in an output that can find the maximum capacity for the augmented path for each transaction. The Reachability graph was generated by the Petri net model for FFA. The Petri net model showed vertices that represent the blockchain as the sink which could include victim transactions. Since the sink has many transactions of the victim and other clients, thus the calculated augmented path is not specifically for the victim transactions. The target was to find the augmented path for the victim transactions only.

[1]A Reachability graph is the brute-force enumeration of all the scenarios in a system [57].

[2]Finding the augmented path for victim transactions that end in the good miners' pool (i.e., added to the blockchain) is covered in Hambolu's dissertation [57], which performs this analysis for the LWM algorithm.

6.6 Summary

In this chapter, we provided the availability analysis results for two preselected-leader consensus algorithms (LWM and BFT-Raft) against malicious miners' DoS attacks. We showed the impact of increasing the number of malicious miners to the system and the ability of the consensus to tolerate malicious miners, which provides the answer to the research question **RQ-5**. And, we provide the answer to the research question **RQ-4** in the second part, which provided an illustration of what methods failed to work to evaluate the availability of chosen consensus algorithms.

CHAPTER 7

CONCLUSION

In this dissertation, we presented a methodology to evaluate blockchain consensus algorithms for liveness and safety. This dissertation offered a novel approach to evaluate the availability of blockchain consensus algorithms and applied it to two preselected-leader consensus algorithms: the LightWeight Mining consensus algorithm (LWM) and BFT-Raft (Tangaroa). This chapter reviews and summarizes the research presented in this dissertation including key results.

7.1 Review of the Dissertation Statement and Research Questions

The methodology presented in this dissertation is designed to address the research questions described in Chapter 1. This dissertation answered the following research questions:

- **RQ-1**: Which consensus algorithms classification(s) (if any exist) is/are better for security evaluation (liveness and safety)?
- **RQ-2**: What are the security requirements that enable a given consensus algorithm for blockchain systems to achieve liveness, safety, and byzantine fault-tolerance (BFT)?
- **RQ-3**: Do the preselected-leader consensus algorithms prioritize both consistency and availability according to the CAP Theorem tradeoffs?
- **RQ-4**: What formal methods are used to test the liveness (availability) of leader-based consensus algorithms (LWM and BFT Raft) for a permissioned blockchain? Which work and which do not?
- **RQ-5**: What is the impact of the number of malicious miners on the liveness (availability) of the blockchain system? How is the system resilient to the malicious miner-based Denial of Service (DoS) attack?

Our dissertation statement, as stated in Chapter 1, is as follows:

1. Provide a common consensus algorithm classification based on how state changes are decided, focusing on the miner-selection process.
2. Identify "security ingredients" and map them to determine the ingredients that enable a given consensus algorithm to achieve liveness, safety, and byzantine fault-tolerance (BFT).
3. Using the Brewer's Theorem (The CAP theorem), analyze the tradeoffs between liveness and safety in terms of availability and consistency, respectively, in consensus algorithms design.
4. Evaluate the preselected-leader algorithms (LWM and BFT-Raft) for liveness (availability) against malicious miner DoS attacks using established techniques and formal methods such as Markov chain and queueing theory.

By answering the five questions above, we addressed the deliverables described in the dissertation statement. The next section recapitulates key results and novel contributions.

7.2 Key Results and Novel Contributions

In chapter 3, we provided a new consensus algorithm classification based on how the state-changes are decided, focusing on the miner-selection and mining processes [2]. State-change-based classification is a way to categorize consensus algorithms based on how the state-changes are decided. A state is a snapshot of the current status of the system and any change in the current variables is considered a system transition to a different state. Examples of system states follow: new transaction(s) are submitted, one miner is offline/busy, a new block is created and broadcasted, etc. Consensus algorithms were classified based on the state-changes because this approach lends itself to different scenarios with better analyses (such as for Petri nets) [97], communication sequential processes [59], queueing theory [69], and Markov modeling [79]. State-change-based classification is a new classification reduced to practice in this dissertation. It is presented for 10 Digital Ledger Technologies' (DLTs') consensus algorithms (Blockchain and Directed Acyclic Graphs (DAGs) [1]) and it classifies DLTs' into two categories: leader- and voting-based consensus algorithm. Leader-based consensus algorithms are further sub-classified into two types: open competition and preselected-leader, where in both there is one miner that is responsible to create and broadcast the block. The winning miner in the open competition category (e.g., Proof of Work (PoW)) is the miner who solves the puzzle first. This can happen by guess, chance, or use of a more powerful machine that solves the puzzle faster. In the preselected-leader consensus algorithms, the miner is preselected based on a specific procedure for that. Voting-based consensus algorithms are further sub-classified into two types: representatives and Gossip voting. Representative-based consensus algorithms are either committee or Practical Byzantine Fault (PBFT) [37] algorithms. In committee-based consensus algorithms, a group of miners is elected as representatives (committee) that are responsible for proposing blocks. In PBFT algorithms, the client waits for a matching reply from a specific number of miners, which make the transaction successful. In PBFT, for the case in which the primary miner fails to respond, miners communicate with each other to confirm the legitimacy of the next miner in line. Gossip voting [1] does not require leaders, followers, or competitions to reach consensus. Instead, it uses the gossip protocol, in which every node randomly picks a neighboring node and sends all the information it knows to this node [2]. This classification provides the answer for the research question **RQ-1**, as shown in Table 3.1 and meet research statement **#1**.

In chapter 4, we compiled the "security ingredients" that enable a given consensus algorithm to achieve safety, liveness, and byzantine fault tolerance (BFT) in both permissioned and permissionless blockchain systems. We provided the organization of these requirements as a new taxonomy that describes the requirements for security. Based on this taxonomy, we deduce that leader-based consensus algorithms prioritize consistency over availability. We asserted that, to improve availability of a given blockchain, algorithm developers need to address the liveness concern by adding steps to ensure the miners-selection process is totally random and fair. The lightweight mining algorithm is a good example of taking steps to improve availability. Overall, the security ingredients and associated taxonomy is a powerful tool for application-specific blockchain protocol designs [3].

Also in Chapter 4, we illustrated the use of Digital signatures to achieve message integrity and non-repudiation in blockchain systems. The CAP Theorem [27] was utilized to explain important tradeoffs between consistency and availability in consensus algorithms design, which are crucial tradeoffs depending on the specific application of a given algorithm. These are contributions of this dissertation, which answer research questions **RQ-2** and **RQ-3** and meet the dissertation statements **#2** & **#3**.

In Chapter 5 and 6, we used proven analytical methods for assessing the preselected-leader consensus algorithm liveness (availability). We presented the availability analysis of the lightweight mining (LWM) algorithm [4] and BFT-Raft against malicious miners' DoS attacks. The queueing theory was employed for the LWM algorithm to identify the average waiting time for client blockchain transactions being targeted by malicious miners. This tested the consensus algorithm's ability to make progress despite the presence of malicious miners or resistant servers. Digital signatures were employed in the protocol to prove the integrity and non-repudiation of messages.

We designed our queue model as a discrete-event simulation (DES) [44] such that the simulation time is advanced based on the event that is closest in occurrence and the simulation can directly advance to the occurrence time of the next event. This novel design enabled the simulation to run in simulated time (jump ahead without waiting for real time to elapse), which made it possible to simulate "longer periods" of time with more transactions. We chose a 95% confidence interval to assure the accuracy as well as the uncertainty (Confidence interval in both linear and log scale) of the estimated average waiting times for our samples to lie within 5%. We applied linear regression to generate the equation that predicts the average waiting time with different numbers of malicious miners.

To verify the validity of the results, we calculated the antilog value of the simulation-results equation 6.1 (which is equal to 10^n, where n is the generated value by the linear equation) to verify the validity of results. We found a strong degree of consistency between both the antilog values and the simulation result values. The design and implementation of this analysis provide the answer to the research questions **RQ-4** and partially meets the dissertation statement **#4**.

The analysis showed that the LWM algorithm is reliable and that it possesses the ability to protect against certain classes of DoS and delay attacks. We showed LWM's ability to continue functioning with almost 90% of the miners acting maliciously in the system. Further, we identified the average waiting time a victim client's transactions take to be added to the blockchain. The analysis also showed that, in its current design, the BFT-Raft algorithm is not byzantine fault-tolerant and it is highly susceptible to malicious miners' attack. In fact, a single malicious miner can halt the system. Both LWM and BFT-Raft guarantee integrity and non-repudiation of messages using a digital signature.

We also provided results on which formal methods succeed in the evaluation such as queueing theory and Markov chain, and which did not work such Ford-Fulkerson algorithm (FFA) [109] analysis. These results provide the answer to the research questions **RQ-4** and **RQ-5** and meet the research statements **#4**.

7.3 Overall Conclusions

We set out to use formal methods to determine the liveness and availability of blockchain protocols, and to systematize the tradeoffs in blockchain design for security. A number of new results were delivered, as summarize above, that key to robustness of blockchains in byzantine circumstances and under DoS-type attack scenarios. Key new results include the classification scheme of DLT Consensus Algorithms and Security-Ingredients Taxonomy (Security) requirements for consensus algorithms to achieve safety, liveness, BFT. We studied the resilience of preselected-leader consensus algorithms against malicious miner threats, and found, for instance, that BFT-Raft has no such resilience unless it adds a verification of miner time interval (which is the basis upon which miners become leader candidates). This was determined as a proof by working with binomial probability distributions and studying the probability of success when even one malicious miner enters the picture. By way of contrast, found that the LWM consensus algorithm can continue to make progress with a high number of malicious miners.

We found that negative result that we cannot determine the percentage time of waiting of victim transactions being added to the blockchain using the Ford-Fulkerson algorithm. Instead, we found that queueing theory offered a viable means to determine this key quantity. In turn, we also tried to apply queueing theory to BFT-Raft, but found that the departure-rate (μ) approached zero; there is no progress and no gain from determining the average waiting time. That is equivalent our determination that BFT-Raft is not resilient to malicious miners, and bolsters that result.

A significant portion of the results described here has been published already in the peer-reviewed literature [2, 3, 4]. Certain results have been extended here beyond those publications, as noted throughout the dissertation.

CHAPTER 8

FUTURE WORK

As future work, I intend to apply security methods and the analysis techniques to different types of blockchains; specifically, the methods are those that address liveness and safety of systems such as availability against DoS attacks, and system consistency against forking. One key goal is to understand the tradeoffs of performance, security, scalability, resource consumption, and transaction rate. I also intend to study the potential of blockchains as defense-in-depth protocols within organizations for managing crucial, but small, data. In this prospective work, I will make use of my newly designed consensus algorithm classification methodology and security ingredients taxonomy to evaluate and test consensus algorithms for liveness and safety used for blockchain systems. Thus, the methodology described here to evaluate availability could prove to be applicable to other blockchain-based systems in which the consensus protocol is probabilistic in nature. This is where the given consensus algorithm uses a miner-selection process to select one miner fairly. Our methodology can be used for these consensus algorithms both to show the reliability and robustness of the system against attacks like those described in this dissertation. Also, we plan to expand the consensus algorithm liveness analysis to other types of blockchain. For instance, we can expand this methodology to permissionless blockchains such as Ethereum (that uses the preselected-leader consensus Proof of Stake (PoS) [110]) and Hyperledger (that uses Proof of Elapsed Time (PoET)).

Additional future work includes the evaluation of other consensus algorithms that have different miner-selection processes using the digital ledger technology (DLTs) State-change-based classification scheme [2], and the security requirements taxonomy [3] for the purpose of expanding and for further generalization with the goal of determining liveness and safety of these blockchains.

In combination with the work done in this dissertation and the above future work, we would subsequently plan to evaluate consensus algorithms for safety (consistency) to assist the overall robustness of these algorithms. Safety requirements are validity, agreement, integrity, and total order [33]. Random communications delay and byzantine failures have the effect of preventing distributed systems from reaching an agreement about the system's global state [33]. Thus, there is a need for either consensus or atomic broadcast protocols [15, 32, 33]. Consensus ensures nodes' agreement on a single transaction whereas atomic broadcast ensures nodes' agreement on a sequence of transactions [18]. Atomic broadcast protocols enable multiple nodes to agree on a value associated with an instance of the protocol. There are many consensus algorithms designed to achieve this task; however, they differ in how they achieve it. This includes design choices such as the number of rounds or phases required to reach a given agreement, the voting procedure to agree on the values, and the decision on the final output [14]. The consensus algorithm safety analysis includes a methodology that addresses all the above safety requirements in addition to the atomic broadcast properties.

Different type of attacks that also could be addressed as future work include the class of malicious miners time-oriented attacks. Consensus algorithms in blockchain systems rely on time synchronicity to coordinate activity among miners. The synchronicity of the underlying network has a direct impact on the correct order of the submitted transactions and also how the agreement among the nodes is reached. Malicious miners may exploit the lack of coordination among the miners and implement their attacks. It is a challenging task to synchronize time in a peer-to-peer blockchain system in the presence of byzantine nodes. Thus, a key aspect of future work would be to evaluate the impact of the underlying network model on the system's time synchronization and its impact on overall security and performance.

Another direction of future work is related to consensus mechanisms in the high performance computing (HPC) area. Our research in the blockchain area has inspired us to study consensus mechanisms in the HPC area. By doing so, we can describe the types of failures that are encountered in HPC systems beyond crash failures because the other failures eventually lead to process failures in HPC. We can classify common consensus mechanisms based on various factors such as the network model, which is important when designing fault tolerant HPC applications. From this, HPC researchers will be able to recognize types of failures and the factors to consider while designing a consensus mechanism and thus be able to detect, mitigate and recover from failures in a more effective and efficient manner. Therefore, future work in this area would be to reimagine fault-tolerant models for Message-Passing Interface (MPI) [81], parallel programs. Such programs would use such data together with generalized faults, not just process and node failures. And, we would plan to use appropriate consensus mechanisms in concert with the faults to be addressed.

REFERENCES

[1] Alexander, R. (2018). *IOTA - Introduction to the Tangle Technology: Everything You Need to Know About the Revolutionary Blockchain Alternative*. Independently published. 34, 35, 82, 83

[2] Altarawneh, A., Herschberg, T., Medury, S., Kandah, F., and Skjellum, A. (2020). Buterin's scalability trilemma viewed through a state-change-based classification for common consensus algorithms. In *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0727–0736. IEEE. xvi, 6, 11, 23, 44, 82, 83, 86, 87

[3] Altarawneh, A. and Skjellum, A. (2020). The security ingredients for correct and byzantine fault-tolerant blockchain consensus algorithms. In *2020 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–9. xiii, xiv, xv, 83, 86, 87

[4] Altarawneh, A., Sun, F., Brooks, R. R., Hambolu, O., Yu, L., and Skjellum, A. (2019). A security analysis of scrybe, a secure-provenance blockchain based on a lightweight consensus mechanism. *Computers & Security*. 5, 18, 31, 84, 86

[5] Ampel, B., Patton, M., and Chen, H. (2019). Performance modeling of hyperledger sawtooth blockchain. In *2019 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 59–61. IEEE. 77

[6] Angelis, S. D. (2020). *Assessing Security and Performances of Consensus algo-rithms for Permissioned Blockchains*. PhD thesis, Sapienza University of Rome. 23

[7] Association, I. R. M. et al. (2018). *Cyber security and threats: concepts, methodologies, tools, and applications*. IGI Global. 45

[8] Athanassiou, P. L. (2016). *Digital Innovation in Financial Services: Legal Challenges and Regulatory Policy Issues*. Kluwer Law International BV. 9, 10

[9] Attiya, H. and Welch, J. (2004). *Distributed computing: fundamentals, simulations, and advanced topics*, volume 19. John Wiley & Sons. 40

[10] Authority, E. (2019). Eos block producer voting statistics. https://eosauthority.com/voting. [Online; Accessed: Nov/25/2019]. 32

[11] Badev, A., Baird, M., Brezinski, T., Chen, C., Ellithorpe, M., Fahy, L., Kargenian, V., Liao, K., Malone, B., Marquardt, J., Mills, D., Ng, W., Ravi, A., and Wang, K. (2016). Distributed ledger technology in payments, clearing, and settlement. *Finance and Economics Discussion Series*, 2016. 9, 10

[12] Bai, Q., Zhou, X., Wang, X., Xu, Y., Wang, X., and Kong, Q. (2019). A deep dive into blockchain selfish mining. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE. 47

[13] Baird, L. (2016). The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. *Swirlds Tech Reports SWIRLDS-TR-2016-01, Tech. Rep.* 34

[14] Baliga, A. (2017). Understanding blockchain consensus models. In *Persistent*, pages 1–18. semanticscholar.org. 14, 15, 16, 41, 88

[15] Bano, S., Sonnino, A., Al-Bassam, M., Azouvi, S., McCorry, P., Meiklejohn, S., and Danezis, G. (2017). Consensus in the age of blockchains. *arXiv preprint arXiv:1711.03936*. 88

[16] Bano, S., Sonnino, A., Al-Bassam, M., Azouvi, S., McCorry, P., Meiklejohn, S., and Danezis, G. (2019). Sok: Consensus in the age of blockchains. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 183–198. 22, 50

[17] Becker, G. (2008). Merkle signature schemes, merkle trees and their cryptanalysis. *Ruhr-University Bochum, Tech. Rep.* 12, 19

[18] Belotti, M., Božić, N., Pujolle, G., and Secci, S. (2019). A vademecum on blockchain technologies: When, which, and how. *IEEE Communications Surveys & Tutorials*, 21(4):3796–3838. 22, 46, 47, 88

[19] Benčić, F. M. and Žarko, I. P. (2018). Distributed ledger technology: Blockchain compared to directed acyclic graph. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1569–1570. IEEE. 11

[20] Bhat, N. B. (2020). *SECURITY ANALYSIS OF BLOCKCHAIN NETWORK*. PhD thesis, Clemson University. 18, 23, 24, 59

[21] Bishop, M. (2003). What is computer security? *IEEE Security & Privacy*, 1(1):67–69. 13, 45

[22] BitcoinBlock (2021). Blockchain technology (bitcoin block data). https://upload.wikimedia.org/wikipedia/commons/5/55/Bitcoin_Block_Data.svg. Accessed: 2020-09-30. xii, 13

[23] Bitcoin.it (2000). Comparison of mining pools. https://en.bitcoin.it/wiki/Comparison_of_mining_pools. [Online, Accessed: Nov/17/2019]. 11

[24] bitshares.org (2019). Delegated proof-of-stake consensus. https://bitshares.org/technology/delegated-\proof-of-stake-consensus/. [Online, Accessed: Nov/17/2019]. 16, 32

[25] Block.one (2018). EOS.IO Technical White Paper v2. https://github.com/EOSIO/Documentation/blob/\master/TechnicalWhitePaper.md. [Online, Accessed: 10/27/2019]. 16, 32

[26] Borowsky, E. and Gafni, E. (1993). Generalized flp impossibility result for t-resilient asynchronous computations. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 91–100. 40

[27] Brewer, E. (2012). Cap twelve years later: How the" rules" have changed. *Computer*, 45(2):23–29. 17, 43, 84

[28] Brooks, R. R., Wang, K., Yu, L., Oakley, J., Skjellum, A., Obeid, J. S., Lenert, L., and Worley, C. (2018). Scrybe: A blockchain ledger for clinical trials. In *IEEE Blockchain in Clinical Trials Forum: Whiteboard challenge winner*, pages 1–2. 19

[29] Brown, R. G., Carlyle, J., Grigg, I., and Hearn, M. (2016). Corda: an introduction. *R3 CEV, August*, 1:15. 14, 41

[30] Cachin, C. (2010). State machine replication with byzantine faults. In *Replication*, pages 169–184. Springer. 16

[31] Cachin, C. et al. (2016). Architecture of the hyperledger blockchain fabric. In *Workshop on distributed cryptocurrencies and consensus ledgers*, volume 310, page 4. 14, 41

[32] Cachin, C., Guerraoui, R., and Rodrigues, L. (2011). *Introduction to reliable and secure distributed programming*. Springer Science & Business Media. 88

[33] Cachin, C. and Vukolić, M. (2017a). Blockchain consensus protocols in the wild. *arXiv preprint arXiv:1707.01873*. xiii, xiv, xv, xvi, 2, 3, 4, 11, 20, 21, 22, 25, 40, 42, 46, 47, 77, 88

[34] Cachin, C. and Vukolić, M. (2017b). Blockchain consensus protocols in the wild. *arXiv preprint arXiv:1707.01873*. 22

[35] Cassandras, C. G. and Lafortune, S. (2009). *Introduction to discrete event systems*. Springer Science & Business Media. 64

[36] Castro, M. and Liskov, B. (1999a). Practical byzantine fault tolerance. In Seltzer, M. I. and Leach, P. J., editors, *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, pages 173–186. USENIX Association. 16

[37] Castro, M. and Liskov, B. (1999b). Practical byzantine fault tolerance. In *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, pages 173–186. 33, 83

[38] Conti, M., Kumar, E. S., Lal, C., and Ruj, S. (2018). A survey on security and privacy issues of bitcoin. *IEEE Communications Surveys & Tutorials*, 20(4):3416–3452. 2, 49

[39] Copeland, C. and Zhong, H. (2016). Tangaroa: a byzantine fault tolerant raft. 5, 20, 21, 30

[40] Cristian, F., Aghili, H., Strong, R., and Dolev, D. (1995). Atomic broadcast: From simple message diffusion to byzantine agreement. *Information and Computation*, 118(1):158–179. 41

[41] David, R. and Alla, H. (1992). Petri nets and grafcet: tools for modelling discrete event systems. *Prentice Hall*. 78

[42] De Angelis, S. (2018). Assessing security and performances of consensus algorithms for permissioned blockchains. *arXiv preprint arXiv:1805.03490*. xi, 15, 38, 42

[43] Delfs, H., Knebl, H., and Knebl, H. (2002). *Introduction to cryptography*, volume 2. Springer. 45

[44] Discrete-Event (2021). Introduction to discrete-event simulation and the simpy language. https://web.cs.ucdavis.edu/~matloff/matloff/public_html/156/PLN/DESimIntro.pdf. Accessed: 2019-12-1. xii, 65, 84

[45] Dwork, C., Lynch, N., and Stockmeyer, L. (1988). Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323. 40

[46] Dwork, Cynthiaand Naor, M. (1993). Pricing via processing or combatting junk mail. In Brickell, E. F., editor, *Advances in Cryptology — CRYPTO' 92*, pages 139–147, Berlin, Heidelberg. Springer Berlin Heidelberg. 28

[47] Ekwall, R. and Schiper, A. (2007). Modeling and validating the performance of atomic broadcast algorithms in high latency networks. In *European Conference on Parallel Processing*, pages 574–586. Springer. 47

[48] Ethereum (2019). Sharding faq. https://github.com/ethereum/wiki/wiki/Sharding-FAQ. [Online, Accessed: Nov/19/2019]. 44

[49] Ferdous, M. S., Jabed, M., Chowdhury, M., Hoque, M. A., and Colman, A. (2020). Blockchain consensus algorithms: A survey. 23

[50] Fischer, M. J. (1983). The consensus problem in unreliable distributed systems (a brief survey). In *International conference on fundamentals of computation theory*, pages 127–140. Springer. 2

[51] Forouzan, B. A. and Mukhopadhyay, D. (2011). *Cryptography and network security (Sie)*. McGraw-Hill Education. 13

[52] Forouzan, B. A. and Mukhopadhyay, D. (2015). *Cryptography and network security*. Mc Graw Hill Education (India) Private Limited. 12

[53] Fotiou, N. and Polyzos, G. C. (2016). Decentralized name-based security for content distribution using blockchains. In *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 415–420. IEEE. 10

[54] Gervais, A., Karame, G. O., Wüst, K., Glykantzis, V., Ritzdorf, H., and Capkun, S. (2016). On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 3–16. ACM. 3

[55] Guin, U., Cui, P., and Skjellum, A. (2018). Ensuring proof-of-authenticity of IoT edge devices using blockchain technology. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1042–1049. IEEE. 19

[56] Gupta, S. S. (2017). *Blockchain*. John Wiley & Sons, Inc. 45

[57] Hambolu, O. (2018). *Maintaining Anonymity and Trust: Applications of Innovative Data Structures*. PhD thesis, Clemson University. 23, 24, 59, 79

[58] Heimerdinger, W. L. and Weinstock, C. B. (1992). A conceptual framework for system fault tolerance. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST. xiv

[59] Hoare, C. A. R. (1978). Communicating sequential processes. *Communications of the ACM*, 21(8):666–677. 26, 82

[60] Hooda, P. (2019). practical byzantine fault tolerance(pbft). https://www.geeksforgeeks.org/practical-byzantine-fault-tolerancepbft. 33, 41

[61] HorizenAcademy (2021). Blockchain vs. directed acyclic graphs (dags). https://academy.horizen.io/horizen/advanced/block-dag/. Accessed: 2010-09-30. xii, 10

[62] Intel (2021). Poet 1.0 specification. https://sawtooth.hyperledger.org/docs/core/releases/latest/architecture/poet.html. [Online, Accessed: Nov/19/2019]. 29

[63] Karame, G. (2016). On the security and scalability of bitcoin's blockchain. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1861–1862. ACM. xv, 29

[64] Katz, J. and Lindell, Y. (2014). *Introduction to modern cryptography*. Chapman and Hall/CRC. 45

[65] King, S. and Nadal, S. (2012). Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, 19. 29, 30

[66] King, S. and Nadal, S. (2017). Ppcoin: peer-to-peer crypto-currency with proof-of-stake (2012). *URL https://peercoin. net/assets/paper/peercoin-paper. pdf.[Online*. 30

[67] Kogias, E. K., Jovanovic, P., Gailly, N., Khoffi, I., Gasser, L., and Ford, B. (2016). Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th {usenix} security symposium ({usenix} security 16)*, pages 279–296. 47, 48

[68] Kokoris-Kogias, E., Jovanovic, P., Gasser, L., Gailly, N., Syta, E., and Ford, B. (2018). Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 583–598. IEEE. 47, 48

[69] Kovalenko, B. G. I. N. (1968). *Introduction to queueing theory*. Israel Program for Scientific Translation, Jerusalem. 26, 82

[70] Lamport, L. (1978). Time, clocks and ordering of events in distributed systems. 21 (7): 558–565. *URL: http://portal. acm. org/citation. cfm*. 42

[71] Lamport, L. (1998). The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2):133–169. 19, 42

[72] Lamport, L. (2019). Time, clocks, and the ordering of events in a distributed system. In *Concurrency: the Works of Leslie Lamport*, pages 179–196. Lamport. xiv, xv, 4, 42, 43, 46

[73] Lamport, L. et al. (2001). Paxos made simple. *ACM Sigact News*, 32(4):18–25. 41

[74] Lamport, L. and Lynch, N. (1989). Chapter on distributed computing. Technical report, Massachusetts Inst of Tech Cambridge Lab for Computer Science. 17

[75] Lamport, L., Shostak, R., and Pease, M. (2019). The byzantine generals problem. In *Concurrency: the Works of Leslie Lamport*, pages 203–226. Lamport. 4, 41, 42, 43, 44, 46, 49, 59

[76] Larimer, D. (2017). Dpos consensus algorithm - the missing white paper. https://steemit.com/dpos/@dantheman/dpos-consensus-algorithm-this-missing-white-paper. [Online, Accessed: Nov/17/2019]. 32

[77] Maria, A. (1997). Introduction to modeling and simulation. In *Winter simulation conference*, volume 29, pages 7–13. 64

[78] Maull, R., Godsiff, P., Mulligan, C., Brown, A., and Kewell, B. (2017). Distributed ledger technology: Applications and implications. *Strategic Change*, 26(5):481–489. 9, 10

[79] Meyer, P., Smythe, R., and Walsh, J. (1972). Birth and death of markov processes. In *Proceedings of the Sixth Berkeley Symposium on Mathematical Statistics and Probability*, volume 3, pages 295–305. 26, 60, 82

[80] Moriggl, P., Asprion, P. M., and Schneider, B. (2021). Blockchain technologies towards data privacy—hyperledger sawtooth as unit of analysis. In *New Trends in Business Information Systems and Technology*, pages 299–313. Springer. 77

[81] MPI Forum (2021). MPI: A Message-Passing Interface Standard. Technical report, Univ. of Tennessee, Knoxville, TN, USA. Note: This is the MPI-4 Final Specification. 89

[82] Mukhopadhyay, U., Skjellum, A., Hambolu, O., Oakley, J., Yu, L., and Brooks, R. (2016). A brief survey of cryptocurrency systems. In *Privacy, Security and Trust (PST), 2016 14th Annual Conference on*, pages 745–752. IEEE. 18

[83] Nakamoto, S. (2009a). Bitcoin: A peer-to-peer electronic cash system. 2, 14, 22, 28, 41, 48, 50

[84] Nakamoto, S. (2009b). Bitcoin: A peer-to-peer electronic cash system. 3

[85] Narayanan, A., Bonneau, J., Felten, E., Miller, A., and Goldfeder, S. (2016). *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press. xiii, xiv, 13, 59

[86] Nguyen, G.-T. and Kim, K. (2018). A survey about consensus algorithms used in blockchain. *Journal of Information processing systems*, 14(1). 11, 22, 23

[87] Nofer, M., Gomber, P., Hinz, O., and Schiereck, D. (2017). Blockchain. *Business & Information Systems Engineering*, 59(3):183–187. 10

[88] Ongaro, D. and Ousterhout, J. (2013). In search of an understandable consensus algorithm (extended version). 16, 19, 20, 30

[89] Ongaro, D. and Ousterhout, J. (2014). In search of an understandable consensus algorithm. In *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*, pages 305–319. 41

[90] Pirretti, M., Zhu, S., Vijaykrishnan, N., McDaniel, P., Kandemir, M., and Brooks, R. (2006). The sleep deprivation attack in sensor networks: Analysis and methods of defense. *International Journal of Distributed Sensor Networks*, 2(3):267–287. 19

[91] Pongnumkul, S., Siripanpornchana, C., and Thajchayapong, S. (2017). Performance analysis of private blockchain platforms in varying workloads. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–6. 50

[92] Popov, S. (2016). The tangle. *cit. on*, page 131. 35

[93] Ross, S. M. (2014). *Introduction to probability models*. Academic press. 63

[94] Ryan, P., Schneider, S. A., Goldsmith, M., and Lowe, G. (2001). *The modelling and analysis of security protocols: the csp approach*. Addison-Wesley Professional. 46, 56, 59

[95] Saaty, T. L. (1961). *Elements of queueing theory: with applications*, volume 34203. McGraw-Hill New York. 61

[96] Saito, K. and Yamada, H. (2016). What's so different about blockchain?—blockchain is a probabilistic state machine. In *2016 IEEE 36th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 168–175. IEEE. 2, 25

[97] Salimifard, K. and Wright, M. (2001). Petri net-based modelling of workflow systems: An overview. *European journal of operational research*, 134(3):664–676. 23, 26, 59, 82

[98] Sankar, L. S., Sindhu, M., and Sethumadhavan, M. (2017). Survey of consensus protocols on blockchain applications. In *2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pages 1–5. IEEE. xiii, 11, 22, 23

[99] Schwartz, D., Youngs, N., Britto, A., et al. (2014). The ripple protocol consensus algorithm. *Ripple Labs Inc White Paper*, 5:8. 15

[100] Seber, G. A. and Lee, A. J. (2012). *Linear regression analysis*, volume 329. John Wiley & Sons. 72

[101] Stuve, E. M. (2004). Estimating and plotting logarithmic error bars. 103

[102] Tanenbaum, A. S. and Van Steen, M. (2007). *Distributed systems: principles and paradigms*. Prentice-Hall. 11

[103] Taylor, A. E. (1955). Advanced calculus. *Blaisdell Publishing*. 63

[104] Van Damme, E. (2012). *Refinements of the Nash equilibrium concept*, volume 219. Springer Science & Business Media. 48

[105] Vranken, H. (2017). Sustainability of bitcoin and blockchains. *Current Opinion in Environmental Sustainability*, 28:1 – 9. Sustainability governance. 3, 29

[106] Vukolić, M. (2015). The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In *International workshop on open problems in network security*, pages 112–125. Springer. 22

[107] Wang, W., Hoang, D. T., Xiong, Z., Niyato, D., Wang, P., Hu, P., and Wen, Y. (2018). A survey on consensus mechanisms and mining management in blockchain networks. *arXiv preprint arXiv:1805.02707*, pages 1–33. 22

[108] wikipedia (2019). Electronic signature. https://en.wikipedia.org/wiki/Electronic_signature. Accessed: 2019-12-1. xii, 45

[109] Wikipedia contributors (2021). Ford–fulkerson algorithm — Wikipedia, the free encyclopedia. [Online; accessed 12-June-2021]. 85

[110] Wood, G. et al. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32. 15, 44, 87

[111] Worley, C., Yu, L., Brooks, R., Oakley, J., Skjellum, A., Altarawneh, A., Medury, S., and Mukhopadhyay, U. (2020). Scrybe: A second-generation blockchain technology with lightweight mining for secure provenance and related applications. In *Blockchain Cybersecurity, Trust and Privacy*, pages 51–67. Springer. 44

[112] Xiao, Y., Zhang, N., Li, J., Lou, W., and Hou, Y. T. (2019). Distributed consensus protocols and algorithms. *Blockchain for Distributed Systems Security*, 25. 40

[113] Xiao, Y., Zhang, N., Lou, W., and Hou, Y. T. (2020). A survey of distributed consensus protocols for blockchain networks. *IEEE Communications Surveys & Tutorials*. 3, 48

[114] Xu, J. J. (2016). Are blockchains immune to all malicious attacks? *Financial Innovation*, 2(1):25. 15

[115] Zhang, R., Xue, R., and Liu, L. (2019). Security and privacy on blockchain. *ACM Computing Surveys (CSUR)*, 52(3):1–34. 43, 50, 51

[116] Zheng, Z., Xie, S., Dai, H., Chen, X., and Wang, H. (2017). An overview of blockchain technology: Architecture, consensus, and future trends. In *2017 IEEE International Congress on Big Data (BigData Congress)*, pages 557–564. IEEE. 10

[117] ZooKeeper (2017). Zookeeper: A distributed coordination service for distributed applications. https://zookeeper.apache.org/doc/r3.5.4-beta/zookeeperOver.html. 16

APPENDIX A

ERROR IN LOGARITHMIC QUANTITIES

The absolute error bar in linear scale is denoted as y and calculated in the equation below:

$$\textbf{95\% Size in linear scale = y} = \frac{1.96 \times \sigma_i}{3\sqrt{5}} \tag{A.1}$$

The absolute error bars in linear scale are symmetric, however plotting these errors in log scale become asymmetric [101]. Thus, we can not use $\log_{10} y$ to plot error bars in log scale, we need to find the delta δ value , which is the change in transforming y from the linear scale to the logarithmic scale. We calculated $\delta \log_{10} y$ as below:

$$\delta \log_{10} y = \frac{d}{dy}\left[\log_{10} y\right] = \frac{d}{dy}\left[\frac{\ln y}{\ln 10}\right] = 0.434 \times \left[\frac{1}{y}\right] \tag{A.2}$$

Using the value of $\delta \log_{10} y$ to plot the error bars, the error size or the 95% size in log scale equation becomes as follow:

$$\textbf{95\% Size in log scale} = y \pm \left(\delta \log_{10} y\right) \tag{A.3}$$

Thus, the final equation to find the error size is given as follow:

$$\textbf{95\% Size in log scale} = y \pm \left(0.434 \times \frac{1}{y}\right) \tag{A.4}$$

In our work, points generated from the equation resulting from the data linear regression are plotted in Figure 6.2 along with the simulation log data, and the error bars are plotted using the Equation A.4, which is illustrated in this section.

APPENDIX B

SIMULATION CODE DESIGN

In our simulation model, we designed the transactions arrivals and departures in a discrete event scheduling fashion using Python. We build two functions, one to generate the transactions arrivals following Poisson distribution. The other function generates the transactions departures following Exponential distribution.

The simulation clock time was manually advanced to the next event without having to wait for the real time advancement, as shown in Figure B.1.

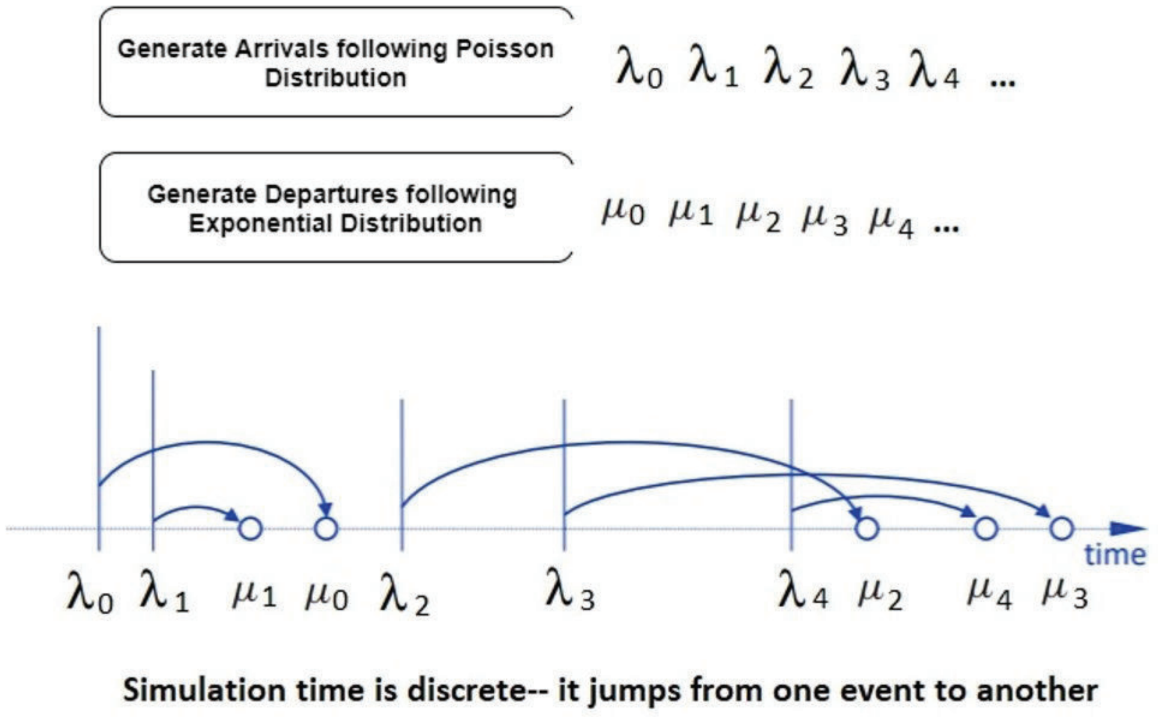

Generate Arrivals following Poisson Distribution
$\lambda_0\ \lambda_1\ \lambda_2\ \lambda_3\ \lambda_4$ ...
Generate Departures following Exponential Distribution
$\mu_0\ \mu_1\ \mu_2\ \mu_3\ \mu_4$ ...
time
λ_0
λ_1
μ_1
μ_0
λ_2
λ_3
λ_4
μ_2
μ_4
μ_3
Simulation time is discrete-- it jumps from one event to another

Figure B.1 Discrete Event Scheduling Logic

This technique helped us run many simulations that would have taken days in real time to run within minutes, below is the Discreet event scheduling (DES) simulation function as implemented in python.

```python
class simulation():

    def __init__(self, L, M):

        self.NumOfTransaction = 1000

        self.ArrivalRate = L
        self.ServiceRate = M
        self.Lambda = 1.0/self.ArrivalRate
        self.Mu = 1.0/self.ServiceRate
        self.TheoreticalMean = 1.0 / (self.ServiceRate
                                         - self.ArrivalRate)

        self.num_in_system = 0

        self.clock = 0.0
        self.arrivalTime = 0.0
        self.departTime = float('inf')

        self.numArrivals = 0
        self.numDeparts = 0
        self.totalWait = 0.0

        self.eventType = ' '
        self.transactionID = 0
        self.minerID = 0
        self.minerType = " "
        self.clientID = 0
        self.clientType = " "

        self.numGoodMiners = G
        self.numBadMiners =  B
        self.numOfMiners = self.numGoodMiners + self.numBadMiners
        self.numClients = V + R
        self.numVictimClients = V
        self.numRegularClients = R

    def Run_transactions(self):

        self.clientIdSelection()
        self.clientIDsList.append(self.clientID)

        V_queue = queue.Queue(maxsize = 1000)
        R_queue = queue.Queue(maxsize = 1000)

        time_event = min(self.arrivalTime, self.departTime)
        self.totalWait += self.num_in_system * (time_event - self
            .clock)

        x = self.minerID in self.BadMinersIDs
        y = self.clientID in self.victimClientsIDs

        if time_event == self.arrivalTime:
            self.eventType = 'Arrival'
            self.transactionID += 1
            if y:
                self.clientType = "victim"
                self.clientTypeList.append(self.clientType)
                V_queue.put(self.transactionID)
                self.clockArrivalList.append(time_event)
                self.V_transactionIDList.append(self.
                   transactionID)
                self.transactionIDList.append(self.transactionID)

            else:
                self.clientType = "Regular"
                self.clientTypeList.append(self.clientType)
                R_queue.put(self.transactionID)
                self.clockArrivalList.append(time_event)
                self.R_transactionIDList.append(self.
                   transactionID)
                self.transactionIDList.append(self.transactionID)
        else:
            self.eventType = 'Departure'
            if x:
                self.minerType = "Bad"
                self.selectedMinerTypeList.append(self.minerType)
                if not (R_queue.empty()):
                    R_queue.get( self.transactionID)
                self.clockDepartList.append(time_event)
            else:
                self.minerType =  "Good"
                self.selectedMinerTypeList.append(self.minerType)
                coin = [0,1]
                if random.choice (coin) == 1:
                    if not (V_queue.empty()):
                        V_queue.get(self.transactionID)
                    self.clockDepartList.append(time_event)
                else:
                    if not (R_queue.empty()):
                        R_queue.get( self.transactionID)
                    self.clockDepartList.append(time_event)

        self.clock = time_event

        if self.arrivalTime <= self.departTime:
            self.handle_arrival_transaction()
        else:
            self.handle_depart_transaction()

    def handle_arrival_transaction(self):
        self.num_in_system += 1
        self.numArrivals += 1
        if self.num_in_system <= 1:
            self.departTime = self.clock + self.generate_service()
        self.arrivalTime = self.clock + self.generate_inerarrivals()

    def handle_depart_transaction(self):
        self.num_in_system -= 1
        self.numDeparts += 1
        if self.num_in_system > 0:
            self.departTime = self.clock + self.generate_service()
        else:
            self.departTime = float('inf')

    def generate_inerarrivals(self):
        b = np.random.exponential(self.Lambda)
        return b

    def generate_service(self):
        a = np.random.exponential(self.Mu)
        return a

    def clientIdSelection(self):
        item =  random.sample(range(1,self.numClients), 1)
        self.clientID = item[0]
        self.clientIDsList.append(self.clientID)
```

VITA

Amani Altarawneh was born and raised in Al Karak, Jordan. She received her Bachelor of Science in Computer Science from Mu'tah University, Al Karak, Jordan. Next, she received her Master of Science in Computer Science from Bridgewater State University in Massachusetts, USA. She joined the PhD program in Computer Science in Auburn University in 2017 and transferred to The University of Tennessee at Chattanooga (UTC) with her advisor Dr. Anthony Skjellum and joined the PhD program of Computational Science /Computer Science in the same year. She worked as a graduate research assistant for Dr. Skjellum, focusing on Cybersecurity and Blockchain Technology. Her research interests include Cybersecurity, IoT and smart cities, High performance computing (HPC), and distributed systems.