DECENTRALIZED SYSTEMS FOR INFORMATION SHARING IN DYNAMIC ENVIRONMENT USING LOCALIZED CONSENSUS

by

Linir Zamir

A Dissertation Submitted to the Faculty of

The College of Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

Florida Atlantic University

Boca Raton, FL

August 2022

Copyright 2022 by Linir Zamir

DECENTRALIZED SYSTEMS FOR INFORMATION SHARING IN DYNAMIC ENVIRONMENT USING LOCALIZED CONSENSUS

by

Linir Zamir

This dissertation was prepared under the direction of the candidate's dissertation advisor, Dr. Mehrdad Nojoumian, Department of Electrical Engineering and Computer Science, and has been approved by the members of his supervisory committee. It was submitted to the faculty of the College of Engineering and Computer Science and was accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

SUPERVISORY COMMITTEE:

Mehrdad Nojoumian (Aug 2, 2022 15:22 EDT)

Mehrdad Nojoumian, Ph.D.
Dissertation Advisor

Mihaela Cardei, Ph.D.

Taghi Khoshgoftaar
Taghi Khoshgoftaar (Aug 2, 2022 18:27 EDT)

Taghi Khoshgoftaar, Ph.D.

Borko Furht (Aug 5, 2022 15:20 EDT)

Borivoje Furht, Ph.D.

Hanqi Zhuang, Ph.D.
Chair, Department of Electrical Engineering and Computer Science

Stella Batalama, Ph.D.
Dean, The College of Engineering and Computer Science

Robert W. Stackman Jr., Ph.D.
Dean, Graduate College

Date

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratitude to my supervisor, Dr. Mehrdad Nojoumian. I am grateful for his continuous support, encouragement, patience, guidance and immense knowledge during the course of my research. I am especially grateful for his insightful comments that greatly improved the quality of my study.

I would like to sincerely thank the defense committee: Dr. Mihaela Cardei, Dr. Taghi Khoshgoftaar and Dr. Borivoje Furht for their helpful insights and guidance throughout each milestone of the Ph.D. journey.

I especially thank my mother for the constant support and encouragement during and beyond the period of this research.

Research was sponsored by the Army Research Office and was accomplished under Grant Number W911NF-18-1-0483. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

ABSTRACT

Author:	Linir Zamir
Title:	Decentralized Systems for Information Sharing in Dynamic Environment Using Localized Consensus
Institution:	Florida Atlantic University
Dissertation Advisor:	Dr. Mehrdad Nojoumian
Degree:	Doctor of Philosophy
Year:	2022

Achieving a consensus among a large number of nodes has always been a challenge for any decentralized system. Consensus algorithms are the building blocks for any decentralized network that is susceptible to malicious activities from authorized and unauthorized nodes. Proof-of-Work is one of the first modern approaches to achieve at least a 51% consensus, and ever since many new consensus algorithms have been introduced with different approaches of consensus achievement. These decentralized systems, also called blockchain systems, have been implemented in many applications such as supply chains, medical industry, and authentication. However, it is mostly used as a cryptocurrency foundation for token exchange. For these systems to operate properly, they are required to be robust, scalable, and secure. This dissertation provides a different approach of using consensus algorithms for allowing information sharing among nodes in a secured fashion while maintaining the security and immutability of the consensus algorithm. The consensus algorithm proposed in this dissertation utilizes a trust parameter to enforce cooperation, i.e., a trust value is assigned to each node and it is monitored to prevent malicious activities over time. This dissertation also proposes a new solution, named localized consensus,

as a method that allows nodes in small groups to achieve consensus on information that is only relevant to that small group of nodes, thus reducing the bandwidth of the system. The proposed models can be practical solutions for immense and highly dynamic environments with validation through trust and reputation values. Application for such localized consensus can be communication among autonomous vehicles where traffic data is relevant to only a small group of vehicles and not the entirety of the system.

DECENTRALIZED SYSTEMS FOR INFORMATION SHARING IN DYNAMIC ENVIRONMENT USING LOCALIZED CONSENSUS

List of Tables x

List of Figures xi

1 Introduction 1

1.1 Overview 1

1.2 Motivation and Contribution 2

1.2.1 Overview 2

1.2.2 Security 4

1.2.3 Information Sharing 4

1.2.4 Localized Consensus 5

1.3 Dissertation Outline 6

2 Background 8

2.1 Preliminaries 8

2.1.1 Blockchain 8

2.1.2 Consensus Algorithm 10

2.1.3 Digital Signature 11

2.2 Blockchain applications for Autonomous Units 11

2.3 Proof-Based Consensus Algorithms 15

2.4 Voting-Based Consensus Algorithms 25

2.5 Attacks and Countermeasures in Blockchain-Based Platforms 30

2.5.1 Mining Attacks 34

2.5.2 Technical Discussion 60
2.5.3 Concluding Remarks 62

3 ISRaft - Decentralized Information Sharing 63
3.1 Introduction 63
3.2 ISRaft: Our New Information Sharing Paradigm 66
3.2.1 Architecture of the ISRaft 68
3.2.2 Leader Election 69
3.2.3 Log Replication 70
3.2.4 Validation 71
3.3 Implementation 72
3.3.1 Implementation Architecture 73
3.4 Analysis of Our ISRaft 74
3.4.1 Comparison 74
3.4.2 Efficiency Analysis 78

4 Consensus Algorithm for Autonomous Units 80
4.1 Introduction 80
4.2 ISRaft Consensus Protocol 81
4.2.1 Design Overview 82
4.2.2 Leader Election 84
4.2.3 Block Generation 85
4.2.4 Data Update and Validation 87
4.3 Experiment 88
4.3.1 ARGoS Simulator 88
4.3.2 Setup 88
4.3.3 Technical Results 89
4.4 Conclusions and Future Work 90

5 Localized State-Change Consensus (LSC) 91
5.1 Introduction 91
5.1.1 Decentralized Autonomous Units 92
5.2 Decentralized Solutions for State Change and Autonomous Systems . 93
5.3 Notations, Definitions, and Properties 95
5.4 Our Proposed Localized State-Change Consensus Mechanism 97
5.4.1 Design Overview 98
5.4.2 Participants 101
5.4.3 Election Process 102
5.4.4 Data Point's State Architecture - Local Consensus 103
5.4.5 Block Generation/Mining 106
5.4.6 Trust and Reputation 109
5.5 Technical Analysis or Our Proposed Solution 111
5.5.1 Effectiveness 111
5.5.2 Scalability - A Storage Analysis 111
5.5.3 Security Analysis 113

6 Conclusions 116
6.1 Future Work 117

Bibliography 119

LIST OF TABLES

2.1 Block Withholding Countermeasures 35

2.2 Selfish Mining Countermeasures 39

2.3 51% Attack Countermeasures 43

2.4 Network Attack Countermeasures 46

2.5 DDoS Attack Countermeasures 52

2.6 Pool Hopping Countermeasures 56

5.1 Notation Used 98

5.2 LSC messages and their reputation costs. 110

LIST OF FIGURES

1.1 Vehicles exchanging information such as vehicle size, position, speed, heading, etc. [18] 6

2.1 Example of a Blockchain network blocks [1] 10

2.2 Example of a Bitcoin block from blockchain explorer 17

2.3 L_3 is a traitor - L_2 point of view [2] 27

2.4 Commander is a traitor [2] 28

2.5 A Blockchain fork [3] 32

2.6 Example of orphan block and stale block [4] 33

2.7 Illustrate of Selfish Mining. M_h is the honest miner, while M_s is the selfish miner 39

2.8 DNS attack. The attacker injects the DNS cache with modified data so when the user queries the server to obtain the IP of peers on the network, he is routed to the malicious network instead of the Bitcoin for example 47

3.1 PRCs that are used in this model. 69

3.2 Data log example. 72

3.3 Comparison of consensus algorithms. 77

3.4 Algorithms latency with different % of malicious servers. 78

4.1 Example of ISRaft blocks. 86

5.1 Leadership term flowchart. 100

5.2 Representation of `L` tree structure 106

5.3 Blocks example in the LSC infrastructure. 109

5.4 Required memory for chains with different `Lt` values. 112

CHAPTER 1
INTRODUCTION

This dissertation proposes two efficient and secured decentralized solutions for achieving a secured data transmission between users without the need for any central authority. Given their decentralized, automated and organized fashion, these solutions can be implemented in an autonomous units environment such as robotic swarms, smart car communications and many more. This chapter covers the motivation and contribution of the two solutions and illustrates the dissertation outline.

1.1 OVERVIEW

Blockchain technology is an immutable append-only ledger storing transactions in a linked chain of individual blocks. Blockchain is one of the very first public data structures that is fundamentally decentralized and is the underlying technology of all cryptocurrencies. With blockchain, it is possible to achieve a consensus over transactions on a distributed immutable ledger without the need of a trusted third party. Trust is achieved by the reliability of the consensus algorithm, which serves as the building blocks of different blockchain applications. Consensus algorithms utilize different cryptography primitives to achieve consensus and trust among the majority of users. Bitcoin, for example, is using the Proof-of-Work consensus and is relying heavily on the SHA-256 hash function. Many also use digital signatures and Diffie-Hellman key exchange.

1.2 MOTIVATION AND CONTRIBUTION

Blockchain technology is mainly used for achieving consensus of transactions over a shared ledger, making it ideal for cryptocurrency for its secured and immutable nature. The primary objective of this work is to use the same blockchain technology to solve the problem of secured and immutable communication in dynamic P2P networks by developing a decentralized consensus protocol that can be implemented in real life scenarios. The idea of using blockchain for information has been studied for some time [5, 6] yet most implementations are using similar preexisted consensus algorithms that can only validate transactions of tokens, currency.

This dissertation presents two unique consensus algorithms designed for secured communication and validation in a dynamic environment with malicious activities. The multi-layer design of these algorithms allow, for the first time, a validation of environmental state, making it ideal for implementation in autonomous units such as robotic swarms, smart vehicles, and more.

1.2.1 Overview

This dissertation addresses the need for decentralized solutions that can be used over a dynamic environment for allowing immutable communication between nodes.

- **Information sharing in the presence of adversarial nodes using raft** [7] paper is the first to propose a decentralized voting-based consensus algorithm for information sharing on the blocks data layer. The proposed algorithm is based on the classic Raft consensus algorithm and it includes trust value and other cryptographic primitives such as digital signatures to achieve information sharing protocol that can handle malicious nodes called ISRaft. This algorithm can be implemented to improve the cooperation among parties in a resource-constrained environment.

- **ISRaft consensus algorithm for autonomous units** [8] builds upon the previous paper, ISRaft for autonomous units improves the original design by reducing the number of functions it requires and improving the overall security of the algorithm. It then presents an implementation of the algorithm using ARoS simulator software that can emulate a communication of autonomous units in a predefined environment. This improved ISRaft is capable of achieving a consensus over the 'states' of different environmental data points. For example, this algorithm can achieve a consensus over the number of blocked roads in a city, a number of high voltage hazards for robotic swarms deployment or other environmental states that are dynamically changed.

- **Localized State-Change Consensus in Immense and Highly Dynamic Environments** [9], presents a new consensus algorithm called 'Localized State-Change', LSC for short, that is capable of achieving consensus by partitioning a set of nodes into subgroups and and achieve what we called a 'localized consensus'. The new algorithm is a voting based algorithm, similar to ISRaft, however it uses localized consensus with validation through reputation value to achieve a majority consensus over a highly dynamic environment. The simplicity of the primitives used for this algorithm also makes it possible to be implemented in resource-constrain devices such as autonomous systems to other devices that communicate on the network. Validation and authenticity of information is achieved by first initiating a local consensus using cryptographic communication means and reputation values. This can then be expanded to a global consensus when the state-change has been verified. LSC assures confidentiality, integrity and validity of messages on the blockchain among all agents.

The remainder of the chapter covers the different aspects of the contributions.

1.2.2 Security

Security is one of the most important aspects of every technology, particularly the technologies that are meant to provide an infrastructure for a public service. Blockchain and cryptocurrencies are known to have security concerns. Different researches have covered various aspects of blockchain and its associated vulnerabilities. Many of the attacks presented in the dissertation, such as selfish mining and block withholding have been studied extensively by many researchers while others such as routing attack and pool hopping have been addressed and discussed more recently. We describe these attacks, provide detection methods and countermeasures and we then utilize this data to measure the security of our proposed consensus protocols. Some of these solutions solely relied on game theoretic models while many others have been examined in a simulated or a real world situation. This dissertation presents two innovative consensus algorithms with security implementations that can withstand most of the covered attacks.

1.2.3 Information Sharing

Setting a communication channel of information sharing among a set of parties has always been a security concern in the research community [10]. With the increasing amount of information that is being shared worldwide as well as the technological capabilities of adversaries, secure information sharing is essential nowadays. Current centralized solutions may provide security for shared data as long as the centralized authority is trusted. This, however, proved to be risky where centralized authorities acted maliciously. The better-known examples are the Facebook scientific experiment [11] and the government surveillance [12]. As such, decentralized solutions and implementations are becoming more common in many domains such as machine learning, supply chain, and information sharing [13, 14, 15, 16]. The proposed consensus algorithms are an improved modified version of the classic Raft consensus algorithm

that allows secure information sharing among a set of parties in the presence of adversaries. The proposed models in this dissertation utilizes cryptographic primitives such digital signature and trust parameters to enforce cooperation [17], i.e., a trust value is assigned to individual nodes to prevent malicious activities over time. This is a practical solution for autonomous units with resource-constrained devices where a regular encrypted communication method can negatively affect the performance of the entire system. This modification intends to further improve the security and capabilities of the proposed consensus algorithms. Achieving a consensus for information can also be referred to as 'information consensus' and it is the underlying technology of different blockchain communication protocols that are used for remote communication, i.e., vehicle to vehicle (V2V) communication protocol. This is a possible implementation of our proposed models as vehicles are getting smarter and smarter and it is important for them to share information among each other such as road condition, traffic and accidents [18]. Figure 1.1 shows an example of a futuristic environment where vehicles and infrastructure can communicate with each other.

1.2.4 Localized Consensus

Localized consensus has been a concern in the blockchain community. Satoshi Nakamoto's invention of 'single CPU per single node' (also called Proof-of-Work) is the most known method for handling a consensus within such a dynamic environment. The 50% majority consensus is achieved by guaranteeing the difficulty of mining new blocks using only brute-force calculations. This method, however, relies on the need of at least 50% of the nodes achieving a consensus, which works great for cryptocurrency and token exchange applications. However, blockchain technology can be the foundation of many other applications such as information sharing and data communications, where achieving 50% consensus is not required. A good example will be terrain recognition for drones. In this scenario, assume a large group of autonomous

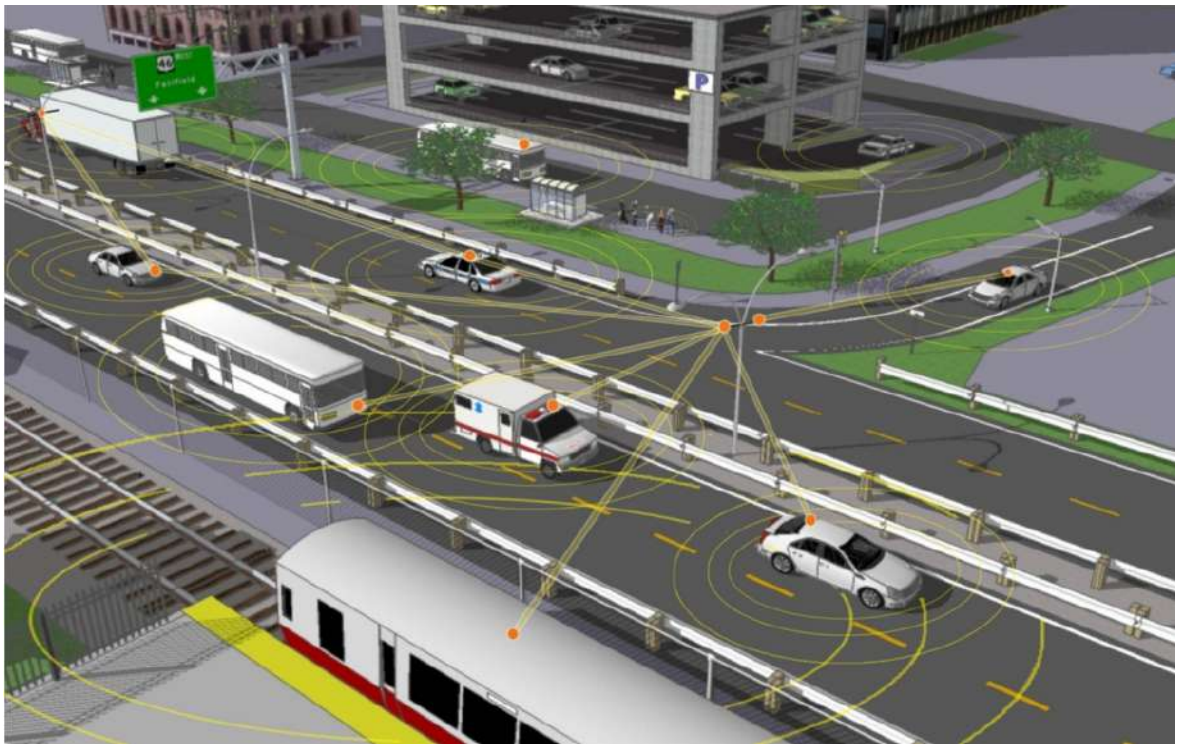


Figure 1.1: Vehicles exchanging information such as vehicle size, position, speed, heading, etc. [18]

drones fly over a large field. At the corner of the terrain there is a large antenna that is a hazard risk for the nearby drones. It is not necessary to achieve consensus among at least 50% of the drones on the presence of the antenna, as it only affects a very small percentage of drones that are flying near it. Thus, a 'localized consensus' is required. This dissertation introduces an innovative way to handle localized consensus in a highly dynamic environment for better communication and information sharing between nodes.

1.3 DISSERTATION OUTLINE

The remainder of this dissertation is organized as follows. Chapter 2 covers the background material of consensus algorithm technologies and methodologies used in this dissertation. Chapter 3 presents ISRaft; the first information sharing consensus

algorithm and briefly discusses the possible implementations. Chapter 4 goes in depth to possible implementations of ISRaft, specifically for Autonomous Units. 5 presents the LSC, the localized-state-change consensus. Chapter 6 concludes the discussion and presents avenues for future research.

CHAPTER 2
BACKGROUND

2.1 PRELIMINARIES

2.1.1 Blockchain

As the name suggests, blockchain is a data-structure that holds data across multiple blocks in a chain chain. This structure was first introduced in 1990 by Stuart Habert and W.Scott Strornetta in [19] as a way to timestamp digital documents. The concept was adopted by Satoshi Nakamoto, who put it into practice to facilitate transactions of the virtual currency known as Bitcoin, where transactions are recorded on a continuously expanding ledger across numerous blocks. Blockchain Technology is based on the idea that each participant, or node, has access to an agreed upon shared ledger by the consensus mechanism methodology. What differentiate blockchains from one another is the underlying consensus algorithm it utilizes. The most common algorithm is the Proof-of-Work (PoW) where nodes are required to 'solve' a unique mathematical problem for them to be the winners thus mining the next block. All consensus algorithms are based on the idea that at least 50% agreement is required for achieving a consensus. Blockchains can have different permission levels [20]. Here are the three common types:

1. **Public Blockchain** - A public Blockchain is permissionless, and anyone can easily participate and validate the transactions. Transactions are public and anonymous in the sense that only a wallet address is linked to any transaction. This type of blockchain is usually maintained by the public community, which means there is a higher level of trust. Bitcoin is the first example for a public

Blockchain.

2. **Private Blockchain** - A private Blockchain is a permissioned Blockchain centralized to one governing organization. Transactions are validated internally and may or may not be publicly readable. Private blockchain requires a registration process for all users, assuring that transactions are backed by the users. These blockchains, however, are considered to be less decentralized as there is an organization that controls a lot of the data and is the one maintaining the network.

3. **Federated/Permissioned Blockchain** - A federated blockchain is a permissioned Blockchain similar to private blockchain, but it operates under the leadership of a group often called the consortium. Predefined consortium nodes control the consensus. The transactions may or may not be public.

The basic idea behind blockchain technology is that it allows parties to send and receive digital assets (Usually referred to as digital currency) on the peer-to-peer network that can then store the transactions on "blocks" in a way that is shared across the network. The user who initiated the transaction, the user receiving it and the transaction data are all registered on this "block", or ledger by using public key encryption and digital signatures. The next step is to validate the transaction on the ledger, and this is done using what is called a "consensus mechanism". The technology of Blockchain may vary but the basic idea remains the same - some consensus mechanism is running in order to mine a new block in a decentralized fashion, while the block is verified by the peers on the network. This technology, even with all of its security achievements, is still exposed to different types of attacks on different levels of its structure.



Figure 2.1: Example of a Blockchain network blocks [1]

2.1.2 Consensus Algorithm

Blockchain solutions do not require a third party trusted authority because they are considered to be a decentralized system. Instead, blockchain uses a consensus method to ensure the accuracy and consistency of data and transactions shared between parties. There are many ways to achieve consensus on a Blockchain network. *Proof-of-Work* (PoW) was presented in a 1993 journal [21]. This mechanism is used by Bitcoin [22] and is the most mainstream Blockchain system today. The mining process is based on electing a 'leader' who will decide the content of the next block. Whenever a new block is mined, the first miner to complete the mining process gets to be the leader of the block. The elected leader is also responsible for broadcasting the block to the network, so that other users (peers) can verify the validity of its content. Motivation for mining is achieved in the form of currency (currency reward, transaction fees). The most widely used proof-of-work scheme is based on SHA-256 and was introduced as a part of Bitcoin and few other cryptocurrencies. This mechanism, however, has a few problems. A big issue is that the mining process depends on the computational power of the leader [23]. Another issue is that miners started to organize in groups called 'mining pools' where they combine their computational power, and then if they create the next block, they distribute the award evenly across everyone in the pool.

Another very popular consensus mechanism in the *Proof-of-Stake* (PoS) [24]. This method was first introduced in 2012 to address the first problem of PoW, which was energy inefficiency. In Proof-of-Stake, every miner's power is determined by the total amount of currency coins he or she has. In this mechanism, an auction is carried out

and whoever wins gets to be the miner. This means that a leader is being selected based on his bid, or how much money he is willing to put 'at stake' in order to win the auction. Unlike PoW, in PoS, whenever a new block is mined, no new coins are being distributed in the system. However, miners are rewarded with transaction fee instead [25].

There are more consensus mechanisms that are being used for different Blockchain networks such as *Delegated-Proof-of-Stake* (DPoS), *Proof-of-Capacity* (PoC), *Proof-of-Elapsed-Time* (PoET).

2.1.3 Digital Signature

Definition 1 *(Digital Signature) A triple of probabilistic polynomial time algorithms* (G, S, V)*:*

- $(pk, sk) \leftarrow G(1^k)$*: Key generator* G *takes a security parameter* 1^k *as input; and returns a secret key* sk *and a public key* pk*.*
- $q \leftarrow S(sk, m)$*: Signing function* S *takes a signing key* sk *and a message* m *as input; and it returns a signature* q*.*
- $0/1 \leftarrow V(pk, m, q)$*: Verify* V *is a deterministic algorithm that takes a public key* pk*, message* m *and a signature* q *as input; and it returns* 1 *if* q *is valid and* 0 *otherwise*

2.2 BLOCKCHAIN APPLICATIONS FOR AUTONOMOUS UNITS

Autonomous Units (AU) co-operation is the idea of self-organization, where its core concept is the formation and development of order between units in a complex dynamic environment. Autonomous units, by definition, operate in a decentralized fashion while making decisions in heterarchical structure (non-hierarchical). The objective of autonomous units is to achieve robustness and flexible coping with dynamics

and complexity. Autonomous Units decision making concerns the possibility of units to be able to change from one state to another. Autonomous units are constantly exposed to malicious cyber attacks and different decentralized solutions have proven to have issues with their scalability correlating to high latency. Also, communication over traditional blockchains can only validate transaction data and not multiple-layers of data shared between different users. As far as we know, there are no consensus algorithms specifically designed to handle communication and validation between autonomous units which can be used as a platform for other decentralized solutions in different fields. This dissertation then proposes a consensus mechanism specifically for achieving state-change consensus between autonomous units while maintaining the security features of the original Raft. This new design can be defined as a **Decentralized Autonomous Units** (DAU) consensus, which we named 'Localized-State Change (LSC) algorithm'. This innovative design is intended to create potentially infinite scalability, and process thousands of state-change transactions per second even with a large number of users in the network. The design is also intended to be implemented in resource-constrained devices, making it ideal for robotic swarms or different DAO applications where users might have power limitations. Arriving at such an algorithm required learning from other similar algorithms of blockchain applications for autonomous units. some of which are:

Research papers [26, 27, 28] aim at finding a solution for reliable communication and decision-making among autonomous units because Byzantine units can have disastrous effects on any autonomous technology. A blockchain approach utilizing Proof-of-Work and other known protocols has been implemented for testing among autonomous agents. By using blockchain, data can be stored in a safe way to prevent unintended or malicious data changes by byzantine units present in the system. Autonomous vehicles, swarm robotics and smart devices are good examples of autonomous units that can benefit from the use of blockchain as a core part of their

infrastructures.

Ferrer et al. [29] highlights the importance of swarm robotics and how the blockchain can be utilized to further improve robotic technologies. Swarm robotics can be seen as a kind of autonomous unit where the characteristics of swarms, such as scalability and resistance to failure, have made swarm robotics appealing to many researchers. However, swarm robotics have some drawbacks. Since swarms mainly communicate through other neighboring robots, there is nothing that allows them to have shared knowledge with other robots in the system. This is where the use of a blockchain is beneficial, where global knowledge can be added to swarm robotics while maintaining local knowledge. By implementing blockchain technologies in swarm robotics, the security of these swarms can also be improved. The blockchain provides reliable and safe communication among robots along with opportunities to validate members of a swarm and prevent malicious attacks. Blockchain also allows for distributed decision-making in swarms. Processes such as voting and leader elections can be implemented into swarms through decision-making that is made possible through the use of a blockchain.

Strobel and Dorigo [30] designs a robotic swarm that defines which color is more prevalent in an environment of both black-and-white tiles. The authors' goal is to determine the frequency of black tiles using a blockchain approach among robots, instead of simply finding which color is more prevalent. Furthermore, a reputation management system is added to manage the presence of byzantine robots. This utilizes a smart contract where each robot's reputation is stored on the blockchain. The reputation of each robot is changed based on how they report the color of a certain tile when the report is compared with other robots' reports in the swarm. When an individual robot reports an incorrect color, its reputation is decreased. When a robot's reputation value becomes very low, its votes are ignored. They then run three separate experiments. The first one is to test if the correct frequency of tiles

could be calculated without using reputation management. The second experiment is to study the effects of the swarm on the blockchain and how efficient it will be. The third experiment is to determine if byzantine robots will have an effect while using the reputation management in the swarm. Overall, the results show that having reputation management is effective in handling byzantine robots at a smaller scale.

Singh et al. [31] present an improved way of decision-making process of swarm robotics by using a blockchain system. Currently, the ways that robots communicate are too centralized and are not resource efficient since blockchains require a large number of resources. In order to improve this, the authors propose to utilize *Proof-of-Authority* (PoA) instead of the more common PoW. In PoA, certain nodes are given the roles of validator. The validators manage and record transactions into the blockchain, with the validator leader gaining the block-publishing priority. This leader is changed after a set period. To verify this approach, a similar method to the colored tiles is used. In a simulated environment, there are grey, white, and black tiles. The robots emit red when on white tiles, green when on black tiles, and do not respond to grey tiles. The objective is to have the robots decide which color tiles emit each color of light. While this test does not consider the byzantine robots, the authors show that it is effective and less resource-intensive compared to the normal blockchain.

Queralta et al. [32] improve data sharing, resource utilization and communication in swarm robotics by using blockchain technologies. They use the PoW consensus protocol to measure the resources the system still has in order to provide proper resource utilization. Moreover, the *Proof-of-Stake* (PoS) is used to validate transactions due to the scalability issues when using PoW. They also use a Single Longevous Blockchain that utilizes ad hoc collaboration in order to allow new nodes to enter the system. This blockchain will also work in situations using permissioned or permissionless blockchains. In addition, a method of ranking is implemented by using

smart contracts, which has resulted in certain robots having higher ranks than others. These rankings are changed based on the needs of the system and will not be recorded in the blockchain itself. Only the data that is validated is stored in the blockchain. Despite the presence of the ranking system, the nodes are not chosen based on their rankings or levels of trust, which provides protection for malicious nodes.

2.3 PROOF-BASED CONSENSUS ALGORITHMS

The next two sections cover a list of some known consensus algorithms separated into two groups; proof-based and voting-based algorithms. The main difference between the two is that the proof-based algorithms rely on providing some mathematical proof to be eligible for mining blocks, while the voting-based algorithms rely on a voting system where the node with the highest votes is usually the eligible miner.

2.3.1 Proof-of-Work

Proof-of-Work, also known as PoW, is perhaps the most popular consensus algorithm that is currently being used by some of the most popular cryptocurrencies such as Bitcoin, Litecoin, Dogecoin and Ethereum (Ethash). Introduced in 2008 by Satoshi Nakamoto [22], Proof-of-Work uses computational power as a way to validate new blocks on the blockchain through a consensus achieving process. Proof of Work utilizes a cryptographic hashing algorithm that generates an almost-unique fixed length output from any given length input and serves as a signature mechanism to authenticate and validate blocks on the chain. When the time comes for a new block to be mined, users on the network compete to find the new block's hash value.

It also serves as the total block signature in the Proof-of-Work consensus mechanism. Each block contains information such as the block header, previous blocks' hash, transactions and the nonce. The goal of the miner is to find the hash value of a block that is smaller than a given value. It can also be seen as setting the first few

bits of the hash value to 0. Finding a small enough hash value is considered to be NP-Hard and can only be solved using brute-force which depends on the availability of the miners computational power. Since the block header and the transactions are publicly available, it also means that a solution is easily verifiable by the other users, also called validators. Proof-of-Work offers a dynamic mining difficulty that is based on the total computational power of the system, also called hashrate. This value changes to maintain an expected block-mining rate per minute. In Bitcoin, for example, the expected block mining rate is roughly one block every 10 minutes. This value is often referred to as the 'block frequency' and together with 'block size', these two values are what determines the speed and size of the total blockchain.

The mining process is based on achieving a consensus between the majority of the users on the network to decide who will be the miner of the next block. This means that when a user finds the correct nonce, he broadcasts it to the other peers on the network who can easily verify that it is the correct value to produce the expected hash value. That user then becomes the leader and gets rewarded with some currency in the form of block reward to motivate them to continue mining. Figure 2.2 shows a sample data of a Bitcoin block.

Limitations of PoW:

This consensus algorithm, however, has some drawbacks. The biggest and most known problem with PoW is the excess power usage. Miners invest in large machines that are capable of solving the hashing problem relatively fast. These machines, however, use lots of electrical power and leaves an environmentally dangerous carbon footprint [33]. What's more, the requirement for powerful machines with high computational powers makes it so that individuals with limited resources might be overshadowed by users with enough resources. This problem leads many individuals to either come up with a solution to this problem or develop alternative decentralized

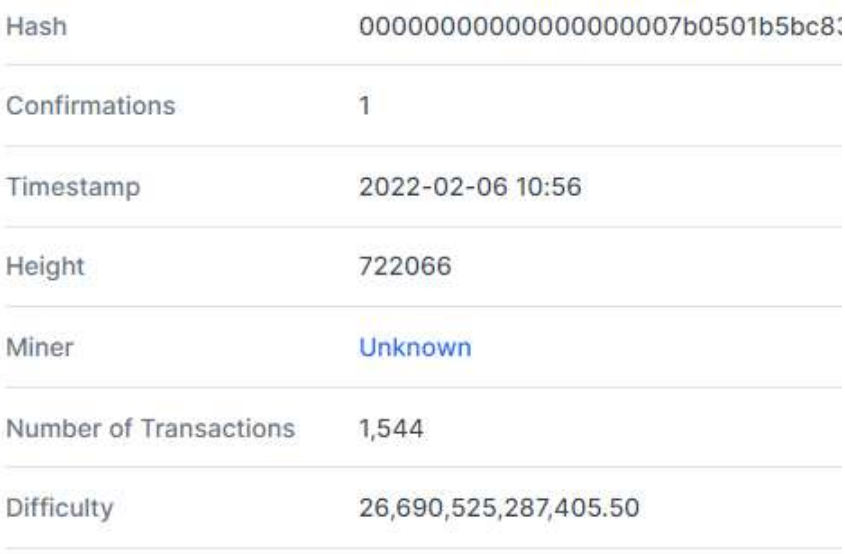


Figure 2.2: Example of a Bitcoin block from blockchain explorer

solutions.

Another problem with the current state of Proof-of-Work is the low transaction speed. Even though this mechanism is considered to be scalable, it has proven to be clustered as more and more users join the network, creating a bottle-neck for transactions. During 2010, worried about possible spam attacks on the Bitcoin network, Satoshi Nakamoto modified the source code of Bitcoin to set a maximum size for each block appended to the blockchain. The value was set to be 1MB. Ever since, the increase in users on the network led to them experiencing delays in minutes and sometimes hours as the transaction rate is over four hundred times larger than the block size limit of 1MB. This problem led to an ever growing debate in the Bitcoin community as to whether to increase the block size limit to allow more transactions per second, or to leave it as is [34]. At the time of writing, the value remained to be 1MB.

2.3.2 Proof-of-Stake

The high energy consumption of Proof-of-Work led to the invention of other consensus algorithms such as Proof-of-Stake (PoS), which aims to lower the overhead costs of network operation. Proof-of-Stake was proposed in 2011 as a replacement to the energy wasting PoW. A year after introduction, it was implemented by King and Nadal [24] in their PPCoin design. PoS utilizes a different approach to mining new blocks where instead of harvesting a high amount of computational power, nodes can stake their coins to lower the difficulty for 'mining' the next block. The stake value of different nodes can be calculated by both the number of coins it adds and how long it holds these coins. When a node stakes the minimum value defined by the system, it becomes a 'Validators' and has the chance of being the miner of the next block. In the PPCoin design, the longer a node holds its coins, the better chance it has of mining the next block using the formula $proofhash < coin\ age\ target$. When a miner is chosen for building the next block, its stake will be cleared and a new round will begin. Validators will lose part of their stake if they approve fraudulent transactions. As long as the stake is higher than what the validator gets from the transaction fees, they will have no reason to act maliciously.

The clear advantage of PoS over other consensus algorithms such as PoW is the reduced required computational power. Given the increased concern in environmental issues, Proof of Stake provides a potentially better outcome for the environment. PoS can therefore reach locations where there is a limitation on the possible acquired computational power, making it more robust and globally available. Another advantage of the PoS algorithm is that miners do not need to calculate a complicated hash function in a tedious process and only need to pass the proof of stake to obtain the mining rights. This in turn reduces the block time and transaction processing time and greatly saves the time for consensus reaching, and the consensus efficiency is significantly improved [35].

There are, however, criticisms of the mechanism, that it is not as decentralized as other consensus algorithms are. PoS requires miners to hold a large number of tokens, something many nodes might not be able to purchase. It can therefore lead to a centralized group of nodes with enough resources to hold a large and ever increasing percentage of the total tokens on the network, thereby reducing the activeness of the entire blockchain. Another criticism is that PoS is more vulnerable to different kinds of attacks such as low-cost bribe attacks. Susceptibility to attacks decreases the overall security of the blockchain [36].

2.3.3 Delegated Proof-of-Stake

Delegated PoS, or DPoS, is a modified version of PoS first introduced in 2014 by Daniel Larimer [37]. In DPoS, nodes on the network vote and elect other nodes called 'delegators' that will secure the network on their behalf. DPoS was developed as an alternative to energy-inefficient consensus of Proof-of-Work blockchains and Proof-of-Stake consensus, that is poorly protected from malicious intentions of stakeholders.

In Delegated Proof of Stake, the stakeholders in the system elect "witnesses" by placing their tokens on the name of their candidate. The coins used that way, much like in regular staking, are not spent but are a representation of the trust stakeholders have with the witness. The more stake a node owns, the more powerful voting he has to assign the witness. The majority of DPoS-based cryptocurrencies don't permit witnesses to prevent transactions, and if a witness misses a block (for instance, because its server went down), it is quickly forwarded to the next active witness.

To manage the system and make fundamental changes, delegates are elected. Delegates manage network settings including transaction fees, block sizes, witness pay, and block intervals but are not in charge of producing blocks or validating transactions. Delegates do not get a salary, but they receive rewards on good behavior and block mining. Their primary incentive to refrain from malevolent activity is fear

of losing money and reputation in the DPoS network. The majority of DPoS-based blockchains consider the magnitude of each stakeholder's stake. The number of tokens a voter has in his possession determines how many votes he can cast. However, there is no rule that forbids nodes from voting because their stake is insufficient. The most democratic method of developing a blockchain consensus algorithm is DPoS, which is distinguished by the fact that every network user has the ability to vote.

The benefits of both PoS and PoW algorithms are combined in the DPoS algorithm. The communication between nodes in the DPoS consensus is faster as a result of the multi-decision maker mechanism with mining rights. The nodes can finish block packaging, broadcasting, and verification quickly, which greatly increases the system transaction speed. Because DPoS doesn't rely on computational resources, it uses less energy. The system's transaction delay and transaction processing speed have both dramatically decreased as a result of the election of decision-makers. At the same time, it allows a large number of nodes to freely enter and quit the blockchain system, making it efficient for a dynamic environment, and the scalability is robust. The DPoS algorithm's drawback is that it depends overly heavily on voting regulations. Poor voting procedures would not only decrease the enthusiasm of participating nodes but also reduce the system's decentralization.

2.3.4 Proof-of-Burn

Proof of Burn (POB) is a consensus algorithm that is also trying to address the main energy consumption problem most traditional algorithms have. The main idea is allowing miners to 'burn' their tokens, and by doing so they are granted the right to mine blocks in proportion to the coins burnt. The way miners can 'burn' their coins is by sending them to an irretrievable secured address. The protocol can count how many coins each user burnt, thus giving users with higher count a higher likelihood in mining the next block.

2.3.5 Proof-of-Space

Proof of Space was originally made as an alternative to the Proof of Work consensus algorithm in public blockchains. Proof of Space(PoSp) was actually proposed as a consensus algorithm even before the creation of Bitcoin. This algorithm works in that there is a prover and a verifier in order to achieve consensus. The verifier essentially authenticates whether or not the prover has the required resources that it claims it has. The resource in Proof of Space algorithms is disk space. In order to store data onto a blockchain in a PoSp algorithm, the verifier needs to know if the prover has enough disk space to store the actual data that the verifier requested. If the prover has enough space, then the data would be stored and appended onto a blockchain. The positives of Proof of Space was that it would be able to reduce energy costs for blockchain networks because only the space that is needed to store data is used, instead of utilizing resources and energy in order to achieve consensus like Proof of Work. However, the issue lies within the fact that for people to be a part of a system that uses PoSp, they would have to sacrifice their personal resources of disk space, which is expensive. Furthermore PoSp would be difficult to employ onto public blockchain networks because every time that a prover is asked by a verifier if they have sufficient resources, they would need to respond, making it a lengthy and confusing process when incorporating larger blockchain systems. Proof of Space may work better on smaller scale blockchain networks.

2.3.6 Proof-of-Identity

Proof of Activity is based upon components of both the Proof of Work and Proof of Stake consensus algorithms. In Proof of Work, authority and decision-making abilities are given to users that are able to solve problems using computational power. Proof of Stake is given to the main stakeholders of a system. Proof of Activity begins when every user in the network uses

their hash power to create an empty block header to eventually be appended onto a blockchain. When any miner creates this block, they then send the header to everyone else in the network. This block header determines random stakeholders for the individual block by using a sub protocol called 'follow-the-Satoshi", essentially where the nodes with the most Satoshi are given higher chances of becoming a stakeholder. Then, the nodes who are currently online in the system validate the block header and check to see if they are one of the stakeholders that were randomly chosen for that individual block. If they are, they then can append the blockchain by adding their transactions onto the block. Every node that discovers themselves to be a stakeholder does the same in the order that they discover they are a stakeholder. In this system, if a selected stakeholder was to be offline then they would not be able to extend the block, and instead the other selected stakeholders would be able to instead. Using the follow-the-Satoshi method in PoA, a user that was trying to control the entire system would have a large number of the total coins that were generated in the system, which makes it safe from large scale attacks. Compared to Proof of Work, if a user had majority of the computational power in a network, they would be able to act maliciously. In Proof Of Activity, the more active users are bound to become more successful since there is a random chance of becoming a stakeholder if you are online. This would be beneficial in systems where high activity is important, since activity implies the enthusiasm and effort that nodes are putting towards the system. The more active nodes would be more deserving of rewards and contributing more to the system. The cons to this system are if a user that has a large amount of satoshi were able to influence the follow-the- satoshi protocol too strongly and was selected as a stakeholder too often.

2.3.7 Proof-of-Importance

The Proof of Importance algorithm aims to fix a fundamental issue in Proof of Stake. In Proof of Stake, If a node has a high stake, they have high odds of being selected to add onto a block and receive rewards. This leads to nodes with high stake winning repeatedly, making the other nodes in the system powerless and less likely to receive rewards. The proof of Importance consensus algorithm was introduced by selecting nodes on a few factors. The two main factors were transaction volume and activity in a system. Using these factors, nodes that were doing the most work and were also demonstrating they were active would be given accounting rights in these systems. In Proof of Importance, the factors would also reset periodically, solving the issue of the same nodes winning repeatedly in consensus algorithms such as proof of stake. However the issue with Proof of Importance is that the algorithm relies very heavily on the free will of the accounting nodes. In the actual selection of the nodes, there is no aspect of randomization, which could cause problems in security. To solve this, a variation of Proof of Importance was made. In this algorithm a dynamic delegation of proof of importance(DPoI) is introduced. This algorithm uses a multitude of factors, other than simply the stake of a node, to select the block leader. These factors aValue, iTrade, Ltime, and credit are used to create the node's iValue, which ranks the node's importance against other nodes. Then a Fibonacci series is used to select nodes at random that have a similar iValue. This algorithm adds more factors to the mix when selecting nodes, but also randomizes the selection, further improving Proof-Of-Importance.

2.3.8 Proof-of-Luck

Proof of Luck is a consensus algorithm that uses trusted execution environments and random number generation to select a leader for consensus in a truly random fashion. Proof of Luck has two functions. The first is PoLRound where the participants of the

consensus plan to start mining on a specific chain by starting from a specific block. Then from that starting block, each participant calls the second function, PoLMine, which allows them to mine their own block. When each node calls the PoLMine function, a random value between zero and one, is generated and assigned to that node. At the end of the consensus round, a select number of the blocks are chosen as winning blocks, and the node that mined the block would receive the reward. In Proof of Luck, the consensus leader is chosen at random, but the way the reward is distributed is also random since the winning blocks are only known at the end of the round. The issue with Proof of Luck is that it was only designed to be used in trusted execution environments, meaning if someone were to have malicious intention inside this environment, they could cause major issues. Proof of Luck can be used in environments where all the nodes have been established to be trustworthy and safe.

2.3.9 Proof-of-Elapsed-Time

PoET is a consensus mechanism algorithm that is often used on the permissioned blockchain networks to decide the mining rights or the block winners on the network. Permissioned blockchain networks are those which require any prospective participant to identify themselves before they are allowed to join. Based on the principle of a fair lottery system where every single node is equally likely to be a winner, the PoET mechanism is based on spreading the chances of a winning fairly across the largest possible number of network participants.

The PoET network consensus mechanism needs to ensure two important factors. First, that the participating nodes genuinely select a time that is indeed random and not a shorter duration chosen purposely by the participants in order to win, and two, the winner has indeed completed the waiting time.

2.4 VOTING-BASED CONSENSUS ALGORITHMS

2.4.1 Paxos

Paxos is a consensus algorithm that is based on achieving an agreement among the majority of users over a synchronized network. Clients can propose a value to the Paxos protocol and a consensus is achieved when the majority of users on the network agrees on the proposed value. Paxos selects a value from all values that were proposed to it and then shares this value to all other users. The underlying protocol is extremely complex and involves assigning different roles to all users on the network. The consensus protocol is abortable, meaning, some processes may abort the running of the consensus if there is a disagreement on the value and a consensus is not achieved. When a client proposes a value to Paxos, it is possible that the proposed value might fail if there was another proposal that already won. The client will then have to propose the value again to another run of the Paxos algorithm. Paxos is widely used and is considered to be the first consensus algorithm that has been proved to be correct. There are many consensus algorithms that are based on it; Raft, ZAB, Fast-Paxos and many others.

2.4.2 Raft

Raft is a consensus algorithm that is based on Paxos consensus but is easier to understand. It offers the same security features as Paxos however it is limited with its ability of handling malicious users. The algorithm was designed in the context of replicated state machines in which the algorithm keeps the logs consistent and identical even when a subset of servers are down. This algorithm performs well while handling servers communications with a client and synchronizing their data. Raft is a consensus algorithm that uses log replication. In an effort to make a more understandable consensus algorithm, it separates the key elements of consensus into

leader election, log replication, and safety. In practice, even a single adversary taking control of a single server would be able to make the protocol unsafe.

2.4.3 Byzantine Generals Problem

The Byzantine General's Problem was first introduced by Lamport, Shostak and Pease in 1982 [2]. This problem refers to an army of generals in which each general commands one part of the army and are situated at distributed locations. The generals can communicate with one another only by messenger, and together the generals must make a common decision whether to attack, retreat or take any other actions. However, some of the generals may be traitors, trying to prevent the loyal generals from reaching agreement. A general can pretend to be the correct one, but present different answers to different generals to manipulate the outcomes. The goal for this problem is to achieve a secured consensus, in which the generals attack as planned by the right generals, and not the traitors. In order to achieve consensus, the commander and every lieutenant must agree on the same decision (for simplicity, attack or retreat).

Definition 2 *Byzantine Generals Problem: A commanding general must send an order to his* $n-1$ *lieutenant generals such that*

- *All loyal lieutenants obey the same order*
- *If the commanding general is loyal, then every loyal lieutenant obeys the order he sends*

The algorithm to reach consensus in this case is based on the value of the majority of the decisions a lieutenant observes. Lamport, Shostak and Pease describe such an algorithm that can reach a consensus as long as 2/3 of the actors are honest. If the traitors are more than 1/3, consensus is not reached, the armies do not coordinate their attack and the enemy wins. Next is the formal definition of the algorithm;

Definition 3 *Algorithm* $OM(0)$

- *The commander sends his value to every lieutenant*
- *Each lieutenant uses the value he receives from the commander, or uses the value RETREAT if he receives no value*

Algorithm $OM(m), m > 0$

1. *The commander sends his value to every lieutenant*
2. *For each* i*, let* v_i *be the value lieutenant* i *receives from the commander, or else he will retreat if he receives no value. Lieutenant* i *acts as the commander in Algorithm* $OM(m-1)$ *and sends the value* v_i *to each of the* $n-2$ *other lieutenants*
3. *For each* i*, and each* $i \neq j$*, let* v_i *be the value lieutenant* i *received from lieutenant* j *in step (2) (Algorithm* $OM(m-1)$*), or else he retreat if he received no such value. Lieutenant* i *uses the value majority* $(v_1, ..., v_{n-1})$

This can be more clear with a visual example. Let C be Commander and L_i be lieutenant i.

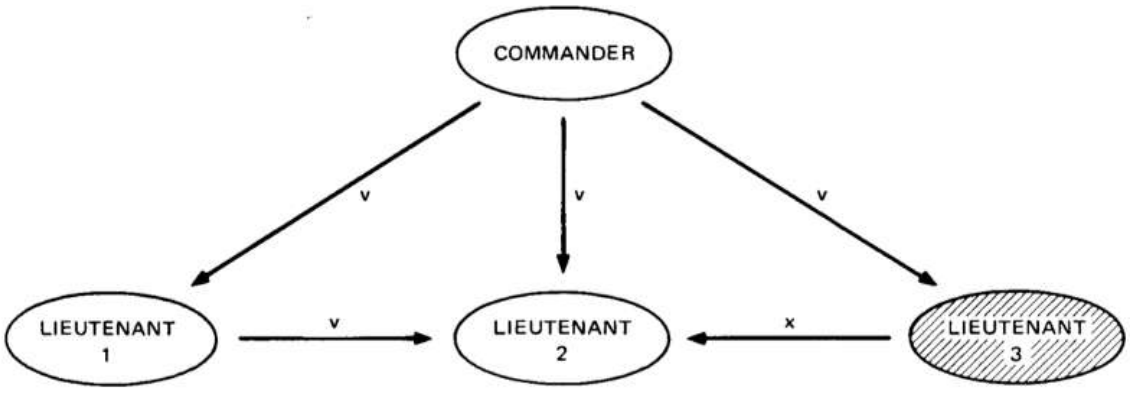


Figure 2.3: L_3 is a traitor - L_2 point of view [2]

Fig. 2.3 shows lieutenant 2's point of view. Following the algorithm, the steps should be:

1. Commander sends a value v to all lieutenants

2. L_1 sends v to L_2, L_3 sends x to L_2

3. L_2 reads the majority of values $(v, v, x) == v$

The final decision is the majority vote from L_1, L_2, L_3 and as a result consensus has been achieved. One very clear distinction we have between this model and Blockchain, is the fact that in blockchain we are dealing with a decentralized system. For that, we can then assume that the commander can act maliciously by sending the wrong value. Fig.2.4 shows the case where the commander is the traitor. Following the

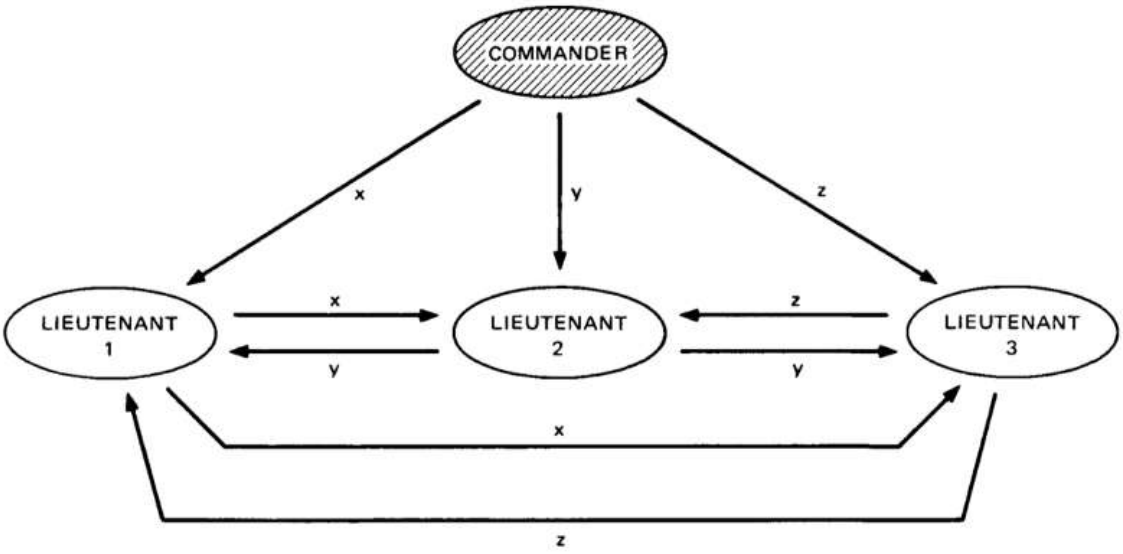


Figure 2.4: Commander is a traitor [2]

algorithm, again, the steps should be:

1. Commander sends x, y, z to L_1, L_2, L_3 respectively

2. L_1 sends x to L_2, L_3 ; L_2 sends y to L_1, L_3 ; L_3 sends z to L_1, L_2

3. $L_1 \leftarrow (x, y, z)$; $L_2 \leftarrow (x, y, z)$; $L_3 \leftarrow (x, y, z)$

They all have the same value and thus consensus is reached. Even if x, y, z are all different, the value of the majority of (x, y, z) is the same for all 3 lieutenants. In the case they are all different commands, we can assume that they act on the default option, **retreat**.

This algorithm, however, is only one possible solution to the problem, and it is one example of the Byzantine Fault Tolerance. Byzantine Fault Tolerance (BFT) is the characteristic which defines a system that tolerates the class of failures that belong to the Byzantine General's Problem [38]. Byzantine Faults are the most severe and difficult to deal with. Byzantine Fault Tolerance has been needed in airplane engine systems, nuclear power plants and pretty much any system whose actions depend on the results of a large amount of sensors. Other variations exist which make solving the problem easier, including the use of digital signatures or by imposing communication restrictions between the peers in the network. Blockchains are decentralized ledges which, by definition, are not controlled by a central authority. Due to the value stored in these ledgers, bad actors have huge economic incentives to try and cause faults. That said, Byzantine Fault Tolerance, and thus a solution to the Byzantine Generals' Problem for Blockchains is much needed.

2.5 ATTACKS AND COUNTERMEASURES IN BLOCKCHAIN-BASED PLATFORMS

Cryptocurrencies such as Bitcoin, Etherum and many others have been implemented using the Blockchain protocol, based on Nakamoto [22]. Like other classical state machine protocols, Blockchain allows participants to agree on a state, in this case, the client balance of a certain cryptocurrency. In this agreed state, the data can reach all other nodes on the network, with no risk of having the data tampered in any way. This technology can be used for more than just crypto-transactions. It can also be used to insure intellectual property, creating and using smart contracts, supply chain track and more. In fact, almost every day there are more and more other industries that manage to implement Blockchain. A new block can be added to the Blockchain in a process called mining. Mining refers to finding a 64 digit hex hash value, namely "nonce", that must be less than or equal to the target hash. Finding this cryptographic hash value can be done in several ways, depending on the Blockchain network. Most common one is the Bitcoin Blockchain that uses Proof-of-Work (PoW) where a miner must solve a challenging mathematical problem in order to acquire the hash value for the next block. This process requires intensive computational work. Once the correct value is found by a miner, it can be easily verified by other miners in the network. A miner who successfully carried out the PoW obtains bitcoins as a reward for his work.

Due to the extremely competitive nature of mining, a miner whose computational power is only a small fraction of the whole network's computational power, has a very low chance to mine a block. Therefore, miners often join mining pools to increase their revenue. Once a mining pool generates a new block, the obtained revenue is distributed among all pool members with the respect to their computational power. The availability of numerous mining pools, brings the opportunity to a miner to switch between pools if he realizes that the new pool can potentially increase his

profit. Lewenberg et al. [39] studied the reward sharing mechanism in mining pools by developing game theoretic models. They concluded that specifically when the rate of transactions is high, it can be very difficult or impossible to keep the distribution of the accumulated revenue stable. Therefore, there is always an incentive for some miners to switch between pools. Mining a new block can also be done using the Proof-of-Stake (PoS) method [24]. This is an alternative to the PoW, which requires a huge amount of energy. In this method, instead of using excessive energy to answer the PoW problem, a PoS mine power is based on the percentage of transactions that is reflective of his or her ownership stake.

With how great and useful this technology is, there are many ways to attack its infrastructure and users. These attacks are hard to counter and to find the origin of these attacks, due to the nature of the Blockchain protocol. There are, however, some implementations that can be made to a Blockchain network to make it possible to get some information on the adversary [40].

Fork

The Blockchain fork can be seen as a collectively agreed upon update of the Blockchain. In this case, the Blockchain splits into two distinct branches. It usually happens as a result of a change in the consensus mechanism, but sometimes it can be unintentionally initialized by mistake as a part of protocol malfunction or issues in the software updates. Looking at Bitcoin, there have been many forks that can be seen as updates to the Blockchain. A very famous one is the Bitcoin Cash fork. Bitcoin original block size is 1MB, which was fine during the first few years, but as Bitcoin gained popularity more and more transactions were initiated and the size started to become an issue. This is when the Bitcoin Cash developed by a group of Bitcoin developers which was a new Bitcoin client with a new block size of 8MB. However, this new

Blockchain was not accepted by the majority of users on the Bitcoin Blockchain, and this was why they created this as a fork on the Bitcoin Blockchain.

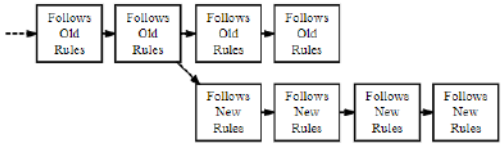


Figure 2.5: A Blockchain fork [3]

Intentional forks can be either soft or hard. A *hard fork* is a *permanent* divergence from the previous version of the Blockchain, blocks of the old version will not be accepted by the newest version. A hard fork is a big change to the protocol that makes previously valid blocks or transactions invalid. Any transaction on the newer fork will not be valid on the older chain. All nodes and miners will have to upgrade to the latest version of the protocol software if they wish to be on the new forked chain.

A *soft fork* can happen in similar situation, however it is common when there is a change in the software protocol, in such a way that it is required to keep the previous transactions, or block, valid to the new rule. It means that the new forked chain will have new rules but it will still work with the old rules (An example will be a change of the consensus mechanism). This type of fork doesn't require everybody in the network to upgrade in order for the new rule to apply, it is possible to have only the majority of nodes in the network. Hard fork, on the other hand, requires (almost) all to upgrade and agree on the new version.

When it comes to resolving forks on a Blockchain network, *soft forks* are relatively easy to fix. Verifying the fork can be done by achieving consensus with all of the peers on the network. This way, the state of the Blockchain can be resumed to its correct state. *Hard forks*, however, are slightly more difficult to resolve because of the need to trace back all the way back to the initial fork. A famous example of resolving

hard fork was made in Spring of 2016 where a Distributed Autonomous Organization (DAO) [41] was created on Ethereum [42]. The basic idea of it was to encourage people to invest and pay Ethers (The currency of Ethereum) to DAO so that they can get the opportunity to vote on and become investors in different projects proposed by Ethereum-based startups. The DAO experiment failed after a hacker managed to steal over $50M USD worth of Ethers. After that, the community of Ethereum voted to return (fork) to the state before the hack.

Stale and Orphaned Blocks

During the mining process, many users attempt to be the leaders of the next block. In a case where more than one miner successfully finds a valid next block at the same time, both proposed blocks are *valid*, but only one can be appended to the Blockchain. At this point, the block that will be verified faster will be the block to be appended. The other block is called *stale block* and will be left on the Blockchain until deleted. There are mining attacks that lead to creation of stale blocks on the network, which makes it so the miner of the block will not be rewarded. Another similar form is the *orphaned block*. As the name suggests, an orphan is a child with no parent. In Bitcoin an orphaned block is a block that is not part of the big chain. Similar to stale blocks, it usually happens when two or more miners solve a block at a similar time. However, orphan blocks are legitimate, verified, and were originally accepted by the network at one point of time. Since they are no longer active and there is no known parent (ancestor) they are rejected from the actual Blockchain. Figure 2.6 shows the

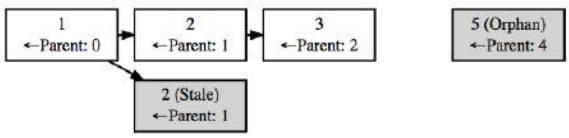


Figure 2.6: Example of orphan block and stale block [4]

difference between a stale block and an orphaned block.

Stale and orphaned blocks are common on most Blockchain networks. In some Blockchain networks, such as Ethereum, this type of blocks can become a part of the Blockchain network. These are called uncle blocks. [43].

Pool Formation

Mining blocks in most Blockchains requires a lot of resources, whether it is computational power, or the currency itself. A single miner usually doesn't hold that many resources alone, leading to the conclusion that it will take years for a single miner to mine a single block[44]. Mining pools is when a group of miners organize themselves to mine a block as a group, or mining pool. Everybody in this group is working together, lending their own resources so that when and if a new block is mined, the group can then share the revenue among themselves. While these pools have been proven to be more useful for a single miner to increase revenue, it leads to some sort of centralization, having these groups more powerful than a single miner joining the network.

2.5.1 Mining Attacks

Cryptocurrencies have been gaining more and more popularity while block mining is becoming more profitable. This results in more miners joining the network to participate in mining competitions. This situation makes cryptocurrencies a reasonable target for adversaries who want to maximize their revenue by exploiting the network and other miners. Through game theoretic analysis [45, 46, 39, 47], it is known that there is an incentive for miners to attack the network in many different ways. In this section we present various kinds of attacks that have been discovered and studied, and their countermeasures.

Countermeasures
Trap trick a dishonest miner into withholding a supposedly full proof-of-work[48]
Special reward for discouraging malicious attacks by providing an additional reward for finding a new block [49]
Distinguishing a PoW from a partially solved PoW [50]
Rational Manager in Bitcoin Mining Pool Dynamic Strategies to incentivize miners not to withhold blocks [3]
ZeroBlock - algorithm incorporates the expected time measurement to validate newly generated blocks[51]

Table 2.1: Block Withholding Countermeasures

Block Withholding Attack

The Block withholding attack was initially proposed by Rosenfeld in 2011 [48] and is relevant to the PoW. The basic idea is that a single miner can decreases the pool revenue by never publishing blocks he mines. Once a new block is mined, participants are then rewarded based on how much effort they had to put towards the solution. The attacker will not be able to obtain anything directly by withholding a block since he cannot submit the proof-of-work independent from the mining pool he is registered with. This is due to a protocol for mining pools that requires only pool administrators to submit a newly generated block [47]. Therefore, this kind of attack will hurt the attacker the same way it will hurt the honest miners. The block withholding attack is performed by the malicious miner for the purpose of sabotage [48]. This is usually conducted by one mining pool against a different one. Eyal [52] used game theory

analysis to show that "no attack" is not a Nash equilibrium. This means that two mining pools attack one another for the purpose of increasing their own revenue. Ironically, the consequence of this strategy is the opposite, both pools gain less if they attack each other. This game theory phenomena is known as prisoner's dilemma. In regards to bitcoin mining, a malicious miner who unofficially and secretly works for a selfish pool namely B, can join pool A for the purpose of sabotaging pool A's revenue. If the malicious miner finds full proof-of-work, he does not reveal it to pool A, nor can he submit it to the bitcoin network to claim the reward for himself. Instead he submits the full proof-of-work to pool B's administrator to prove the sabotage he caused to pool A and receives award from pool B. Bag et al. [50] shows that there is an incentive for a malicious miner to attack the larger pool and receives award from the smaller pool. Block withholding can also be a prelude to selfish mining attack, which will be discussed in the next part.

Detection and Countermeasure: Block Withholding within a mining pool can be detected by relying on the theory of probability. For every pool, the ratio between the computational power of the mining pool and the whole network's computational power represents the probability of finding the full proof-of-work by that mining pool. If the real world number is significantly below the theoretical number, there is a high possibility that the pool is the victim of Block Withholding attack. Even though it is not difficult to detect such an attack, finding the source of the attack could be challenging [53]. Rosenfeld [48] suggests a trickery that can be employed by the pool manager to dupe a dishonest miner into withholding a supposedly full proof-of-work. Pools can catch dishonest miners by incorporating this method at the expense of dedicating a portion of their computational power on a task that does not produce any potential revenue. As discussed earlier, withholding a full proof-of-work alone does not benefit the perpetrator and often this kind of attack is just the tip of the iceberg. Block Withholding might be the sign of a deeper and more compli-

cated adversary activity. Since the Block Withholding attack can be conducted for different purposes, researchers have suggested various countermeasures based on both the attacker's incentive and the complexity of the operation. Bag and Sakurai [49] consider a model in which a miner from a mining pool launches a Block withholding attack on a target mining pool. Then the authors investigate the parameters that could increase the profit of the attacker and finally they propose a rewarding scheme called "special reward". The goal of this new scheme is to discourage the attackers by rewarding them additionally if they find a new block. In addition a block withholder who never submits a full proof-of-work will be denied from this reward and as a result, the attack is not profitable anymore.

Bag et al. [50] presents an attack model in which an independent miner is hired by a malicious pool manager to join a target rival mining pool. The hired miner supposedly works for the target mining pool. But if he finds a full-proof-of-work, he withholds it from the target mining pool and instead he informs his employer about the proof-of-work and receives rewards in return. Then the authors provide a strategy to discourage such an attack. This strategy which, can be implemented using hash functions will prevents the miners to distinguish a full proof-of-work from a partial proof-of-work while making the pool administrator to request a verification from the miners that shows if they had found a full proof-of-work. Lee and Kim [54] present a method to prevent Block withholding attacks by detecting the infiltration first. The authors argue that the attack is the result of infiltration. Therefore, in order to combat the attack effectively, the focus must be on infiltration detection. A pool that is suspicious of being attacked by another pool, can infiltrate into the attacker pool and investigate the attack. If the attack is confirmed, it eliminates the damage by reducing the revenue shared by the attacker pool.

Solat and Potop-Butucaru [51] proposed a new algorithm to prevent intentional Block Withholding attack. This new algorithm namely, ZeroBlock, incorporates the

expected time measurement to validate newly generated blocks. For every transaction, an expected time is considered and calculated locally by the nodes. The expected time depends on the size of the network as well as the difficulty level for solving proof-of-work and therefore predictable. If the adversarial node does not submit its block before the calculated expected time, the block becomes invalid and will be disregarded by honest nodes. The authors state that, by implementing this solution, it is not possible to create a fork of a blockchain intentionally.

A relatively new approach for countermeasure arrives from the study of rational pool managers. Feifan et al. [3] present a model where miners are modeled into short-time-reward-oriented and long-time-reward-oriented. It claims to be a more realistic, reasonable and comprehensive assumption about the decision of miners when compared to other research. A manager of a mine pool can incentivize miners in the pool with not too large hash power and gain extra rewards at the same time with enough hash power and network connection advantages. The hash power and network connection advantages it requires for the attack to be profitable are lower than that of a selfish mining attack, and the reward expectation is higher. The manager may apply this strategy of whether to withhold and discard other miners blocks in more advanced and comprehensive attacks such as stubborn mining to get more extra rewards.

Selfish Mining Attack

In the honest mining strategy, as explained before, if a miner finds the full proof-of-work for the current block in the Blockchain, he then publishes the newly generated block publicly while earning the reward for solving the problem. Consequently, all other miners would stop solving the already solved hash and move on to the next block.

Selfish Mining is the act in which a dishonest miner finds the full proof-of-work for the block, but he withholds the solution from the public. Similar to the block

Countermeasures
Adjusting to consensus for making sure that the maximum mining power is smaller than 25% of the total mining power[55]
Freshness Preferred - An extension with unforgettable timestamp to make it hard to form a group of selfish mining[56]
Revising the fork to promotes the blocks that are linked to the competing blocks of their predecessor [57]
ByzCoin - Designed to increase the security and consistency of the Blockchain based systems[58]

Table 2.2: Selfish Mining Countermeasures

withholding attack only here the dishonest miner continues the mining process generating as many more blocks he can, before someone on the network is close to solving the hash. Figure 2.7 shows an example of the selfish mining attack, with the selfish miner, namely M_s, mining the linked blocks before anyone else can by not releasing the full proof-of-work to the public.

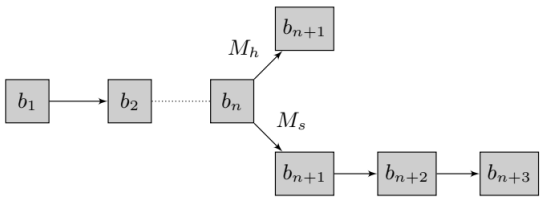


Figure 2.7: Illustrate of Selfish Mining. M_h is the honest miner, while M_s is the selfish miner

The effort put by M_h will be completely wasted due to the selfish miner M_s, and

this is the key of this strategy. Selfish miners want to force honest miners to do wasted computations, specifically on the stale public branch. These selfish miners basically wish to have honest miners spend their time and resources on mining blocks that are already known to not be a part of Blockchain. After the selfish miner releases the solved block, he already has the full proof-of-work for the other blocks he managed to mine, and so he gets a reward for solving these blocks as well. It is clear that there exists an incentive for using the selfish mining strategy [59, 60].

The idea of a selfish mining strategy can cause substantial risk to the health and the reliability of the bitcoin network. In the original bitcoin paper, Satoshi [22] claimed that the bitcoin network is secure as long as the computational power of the honest miners remains the majority. However, according to the more recent studies, it has been proven that the selfish miners can still undermine the security and the integrity of the bitcoin network even if they don't control the majority of the computational power. In particular, Eyal and Sirer [55] show that in order for a pool to conduct a profitable selfish mining attack, it only needs more than a quarter of the total computational power. In a different paper, Sapirshtein et al. [61] analyzed the minimum resources required to conduct a selfish mining attack namely 'optimal selfish mining strategy'. This malicious mining strategy will remain at least as profitable as honest mining which subverts the security of the bitcoin network.

Intermittent selfish mining [62] is another form of the selfish mining attack that consists in alternating selfish and honest mining during consecutive difficult periods. The idea is to fully profit from the decrease of the difficulty but the downside is that the difficulty does not stabilize and recovers after each phase of honest mining.

It should be noted that the honest miners' block is not destroyed; rather, it leads to it becoming a *stale block* as explained in the previous section. This attack can also lead to a *fork* in the chain, in a case where there are two miners who compete to mine their block and add it to the network. When this type of fork happens, it can cause

some sort of a delay in the network, which might lead to future possible attacks.

Cyril et al.[63] define 'profit lag' as the minimum time it takes to be profitable for a dishonest miner using a selfish mining attack. It is the difference between the average revenue of selfish and honest mining, where if a selfish miner starts executing his attack, he can, at any moment, get profits from these attacks. However, if the miner wants to repeat the attack cycle, he won't be able to make the same profit.

Detection and Countermeasure: There are not many ways to detect a selfish miner. One way to detect is to look for a pool with many stale blocks, for all the blocks that were solved but never linked to the network because of the selfish miner. However finding stale blocks is not easy due to the nature of these blocks, being isolated and not connected to blocks on the chain.

As for countermeasure selfish mining attacks, Eyal and Sirer [55] suggested an adjustment to Bitcoin consensus to reduce the chance of having a selfish miner in the pool. The new adjustment includes a randomized mechanism that prevents mining pools from conducting a profitable selfish mining strategy. This mechanism would work effectively as long as the maximum mining power of the pools is smaller than 25% of the total mining power. In addition, the prevention of the selfish mining strategy requires at least 2/3 of the miners to be honest.

An extension to the countermeasure presented in [56] by Heilman, which introduced a new concept called "Freshness Preferred (FP)". The new defense increases the minimum share of mining power from 25% to 32%. The new solution incorporates the use of unforgettable timestamps and it is also robust to the compromises. The author also shows that it will be difficult for a selfish miner group to work against this defense mechanism. This is due to the situation that a member of the selfish mining group is able to anonymously reveal the fact that the group has compromised the system. This ability gives the members an incentive to blackmail other dishonest miners. Therefore, it will be difficult and risky for selfish miners to form a group.

Zhang and Preneel [57] state that the solutions suggested by [55] and [56] are only effective if the selfish chain is shorter than the public chain and as the result, these defenses are incapable of combating a resourceful selfish miner. Then the author proposes another solution that is effective even if the selfish miner's chain is longer than the public's chain. This defense solution is based on revising the fork resolving policy that dismisses blocks that are not submitted on time and instead, promotes the blocks that are linked to the competing blocks of their predecessors. According to the authors, this solution outperforms the previous solutions suggested by [55] and [56] and it is also backward compatible.

The solution introduced in [51] as a defense against block withholding attack, is also suggested as an effective and practical solution to address the selfish mining problem. Kokoris-Kogias et al. [58] introduces a new algorithm called "ByzCoin" which is claimed to be Byzantine fault tolerant. [2] This algorithm is designed to increase the security and consistency of the Blockchain based systems such as Bitcoin. The authors suggest that by equipping Bitcoin with the ByzCoin algorithm, the selfish mining strategy becomes ineffective. The algorithm resolves forks instantly, making a private fork a waste of time, revenue, and resources

51% Attack

51% attack is an attack that happens when an adversary has the majority of the network's mining power (i.e hashrate). This adversary can be a single miner or a group of miners in a mining pool. This can cause many issues such as prevention in transactions or blocks from being verified, conducting a double spending attack [68], zero-confirmation transactions [69], blacklisting and censoring nodes [70], ETC. An example for such attack happened in 2014, when then mining pool GHash.io had the majority of the mining attack for one day on the Bitcoin network [71]. GHash.io later shrunk in size and was eventually closed in October 2016.

Countermeasures
Two-phase PoW - Continuous-Time-Markov-Chain to prevent large pools from creating a hegemony[64]
Machine Learning rules to detect and stop suspected attackers by monitoring the stakeholders on the Blockchain network[65]
Block Maturity Level (BML) - Requires the miner to mine at least one block to add it to the Blockchain
Random Mining Group Selection - Split miners to multiple groups, then chose one group randomly for participating in the next mining competition[66]
Verification process for joining mining pools with strict rules[67]

Table 2.3: 51% Attack Countermeasures

The 51% attack is considered as the worse case attack scenario that could disrupt the security of the Bitcoin so severely that the honest members will eventually leave the network. In [72] the authors argue that no party within the network will gain a long term benefit from bitcoin by conducting such attacks. This is due to the fact that the value of any currency is the product of the public trust in that currency. The security threats caused by the majority attack, can ruin the public confidence in Bitcoin and therefore the collapse of Bitcoin will be inevitable. Therefore, there is no incentive for Bitcoin miners to attack the network in this way. However, this argument does not dismiss the possibility that a party who has an incentive outside of the bitcoin network, can conduct the 51% attack for the purpose of destroying the Bitcoin's economy. This kind of attack is known as the Goldfinger attack. Governments or

other organizations may have an incentive to conduct a Goldfinger attack against the Bitcoin network.

Adversaries do not always need the majority of the Blockchain networks mining power to carry out the attack. There were cases of this majority attack that involved the adversary having only a third of the total mining power on the network, yet it still leads to undesired results. A Blockchain application that is specifically designed for IOT (Internet of Things), known as "The Tangle" [73] can be compromised, theoretically, with much lower hash power. Bahack et al. [74] showed that this type of attack is possible to achieve with only 1/4 of the network's computational power.

Detection and Countermeasure: These attacks have been widely discussed, and have many countermeasure proposals to reduce the chances of a monopoly in the Blockchain networks.

Bastiaan [64] introduced the concept of "two phase proof-of-work" (2P-PoW) to counter these attacks. This newly developed method is a model in which the miners are required to solve two challenges instead of a single one on a classic Blockchain network. 2P-PoW prevents adversaries from executing coordinated attacks.

Dey [65] proposed a methodology in which the activity of stakeholders in the Blockchain network can be monitored by using an intelligent software agent. This software agent can detect the majority attack by relying on supervised machine learning algorithms. If the attack is likely to happen, the system implements the set of rules to prevent the Blockchain confirmation from the attacker.

Memon et al. [75] introduce a new consensus protocol called Block Maturity Level (BML) for Blockchain related technologies. The authors stated that by adapting this new protocol, the attacker would need more than 51% hash power to undermine the network. This new protocol requires the miner to mine more than just one block in order to add the first block onto the Blockchain. Once the miner finds the first block, the status of the block remains "pending" until the miner adds 4-5 sub-blocks

to the original block. The approximate time for the sub-blocks will be a quarter of the time of the original block. Once these extra sub-blocks are added, the status will change from "pending" to "active". BML protocol decreases the risk of 51% attack significantly. The attacker needs to mine all the subsequent sub-blocks which makes the attack more difficult to succeed.

Bae et al. [66] propose a new technique called Random Mining Group Selection. The proposed approach splits miners onto multiple groups then selects one group randomly. Only the peers from the selected mining group are allowed to participate in the mining competition for the next block. Each peer can find its mining group through a hash function and its wallet address. Once the new block is generated and broadcasted over the network, the rest of the miners can examine whether the new block is added by a miner from the selected mining group or not. This can be done by comparing the hash value of the previous block to the block creator's address. This procedure can defend the network against 51% attack effectively. By increasing the number of groups, the chance of attack will be decreased significantly since the groups are selected randomly. This mechanism reduces the block mining hash power to the hash power of the selected group since the peers outside of the selected group don't participate in mining. Also, the difficulty level of mining will be adjusted to the size of the mining group accordingly.

Bala and Manoharan [67] suggest a method of criteria checking for miners to participate in the mining process as well as a verification process for joining mining pools. This proposed method includes a number of rules that are designed to evaluate the trustworthiness of the Blockchain users. The verification process is aimed to prevent malicious clients from joining the mining pools. The rules include:

1. Check the history of the miners to determine if they have involved been in an attack

2. Check the Bitcoin balance in their wallet

3. Determine if the miners hash power is added to the pool's hash power which the miners are joining, the sum does not exceed more than 50% of total hash power.

Network Attacks

Countermeasures
Blacklisting IP addresses of users with low trust score, which is based on the behavior on the network[76]
Short term countermeasures: Increase diversity, monitoring round-trip time (RTT), monitoring user statistics, etc.. **Long term countermeasures:** : Encrypting Bitcoin communication, using MAC, using UDP heartbeats, etc.. [77]
Counting the number of stale blocks and comparing it with the baseline [78]

Table 2.4: Network Attack Countermeasures

These attacks are somewhat different than the other mining attacks that were covered in the previous sections in the sense that these attacks are associated with the peer-to-peer network, where the goal of the adversary is to block miners and other nodes from the network, to limit their access to the network or to enforce conflicts between the nodes. The attacks include among others the DNS attack and Eclipse attacks which will be covered next. **DNS Attacks**

Domain Name System is the name of the service used to identify different IP addresses using their domain name. On the Blockchain network, when a new user joins for the first time, he needs to discover the other active peers which are identified

by their IP addresses. This process can be done with DNS where the DNS seeds are initated by different nodes once they join the network.

The developer's guide of Bitcoin [79] explains that the DNS is vulnerable to all kinds of attacks, cache poisoning and more. This makes the DNS a great place for adversaries to execute their attacks and isolate Blockchain peers. In this attack, just like a regular DNS attack, the adversary is tampering with the users' DNS service by injecting an invalid list of seeder nodes or poisoning the DNS cache. What it all means is that an attacker can inject a fake list of seeders, he will potentially be able to isolate users on the Blockchain and lead them to a compromised network. Figure 2.8

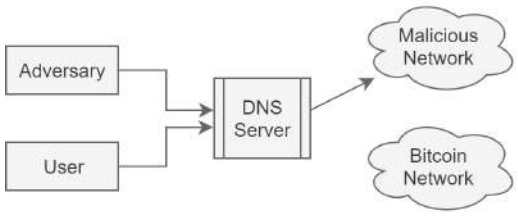


Figure 2.8: DNS attack. The attacker injects the DNS cache with modified data so when the user queries the server to obtain the IP of peers on the network, he is routed to the malicious network instead of the Bitcoin for example

shows an illustration on how such a DNS attack can be carried out by modifying the DNS cache. An adversary can inject false information inside the DNS server so that when a user is queries the DNS for information about the Blockchain network (active peers, transaction, etc.) the DNS service directs the user to the malicious network, where the attacker can 'feed' the user false information and make him vulnerable to different attacks. **Eclipse Attacks**

Each peer on the Blockchain network is identified directly by its IP address. Communication with other peers is established by using a randomized algorithm that finds 8 random peers on the network and forms long lived connections. For the purpose of establishing the connection (handshake) a maximum number of 117 unsolicited incoming connections are allowed. Through this communication, peers can broadcast

and receive important data on the network, i.e the latest state of the Blockchain in the network.

This network communication protocol can successfully provide an open and decentralized network environment. But there is a crucial security trade-off. The malicious peers can also join the peer-to-peer network with the intention of abusing honest peers or undermining the security of the network. An attacker can target a victim node from the network and then isolate its connections to only the IP addresses that are controlled by the attacker. This is the basic idea of the Eclipse attack [80, 81, 76], where the attacker can manipulate the victim's connections on the network.

Eclipse attack often conducted for different purposes including Delivery-tampering attacks [82] engineering block races, double spending, generating a hidden fork on the network, making the victims to waste their computational power on old or obsolete blocks and finally hijacking the victims computational power. The minimum required resources to conduct an Eclipse attack has been the subject of much research.

Marcus et al. [80] presented an eclipse attack model on Ethereum network that can be imposed by an adversary who controls only two IP addresses on two computers. Heilman et al. [76] used a mathematical model to demonstrate an eclipse attack, then to confirm the practicality of this model, they performed it on their own bitcoin. They found that an attacker with only 32 distinct IP addresses or a 4600-node botnet, is able to conduct an eclipse attack against a victim with at least 85% chance of success.

Routing Attacks

This attack was first introduced by Apostolaki et al. [77] in 2017. Routing attack can be conducted in small-scale targeting individual peers or large-scale targeting the whole network. The authors describe two kinds of routing attack individually. The first kind is called *Partitioning Attack*.

By conducting this attack, the attacker aims is to disconnect nodes or sets of nodes from the network completely. To do so, the attacker needs to disconnect all the

connections between the victim nodes and the rest of the network. Due to the complicated structure of the Bitcoin network, the initial isolation may not be complete and the isolated nodes may still leak and be able to hold some communications with the rest of the network. The attacker is able to detect and eliminate these leaks until the victim nodes are completely isolated and the network is partitioned. The second kind is called *Delay Attack*. The purpose of this attack is to slow down the propagation of blocks sent from or received by the victim nodes. In contrast to partition attacks, which requires a perfect isolation, delay attacks can be conducted by just intercepting a subset of connections between the victim nodes and the rest of the network. The detection of delay attacks requires more challenges than partition attacks. Routing attacks can cause a significant loss to mining pools and individual miners. By partitioning the network or interrupting the propagation of blocks, the attacker can force the victim miners to waste their mining power on the blocks that are eventually discarded.

Khalilov et al. [83] further studied various partitioning attacks on Bitcoin. The authors show that the Bitcoin network is getting more and more centralized at the AS-level. Through data collection and analysis, the authors show that the consensus among Bitcoin peers is non-uniform. As a result, the network is becoming more vulnerable to partitioning attacks. The authors then demonstrate four variations of partitioning attack: spatial, temporal, spatio-temporal, and logical. **Stubborn Mining Attack** In general, the selfish mining strategy is used by the attacker to acquire short term rewards. From the attacker's point of view, if his private chain is longer than the public's chain, he will continue mining on his private chain for the hope of generating even more blocks. But if the attacker's private fork falls behind the public chain, he disregards his private fork and reverts back to the public chain. Stubborn mining, first introduced by Nayak et al. [81], is a selfish mining strategy, when the attacker persists in selfish mining. So in general a stubborn miner

will not give up on selfish mining easily. The authors argue that the selfish mining strategy is not optimal. It means that by following the stubborn strategies, the attacker will be able to gain even more rewards than if he only employs selfish mining strategies. Then the authors introduce three different stubborn strategies namely, "Lead stubborn", "Equal Fork stubborn" and "Trail stubborn". In these strategies, the attacker strategically reveals his hidden individual blocks based on the current state of the Blockchain rather than just revealing his entire fork at once. This allows the attacker to increase his revenue by up to 25%. The authors also studied a model in which, stubborn mining is combined with eclipse attack, hence the attacker can gain up to 30% more rewards in comparison with the less advanced use of eclipse attack. The profitability of stubborn mining strategies, depends on the duration of the attack and the analysis of the expected revenue [84].

Detection and Countermeasure: Detecting and countering network attacks is slightly different than other mining attacks, because that the nature of these attacks in to tamper with the network of a specific user and not with the Blockchain network. This means that there are many options for attackers to achieve their goals but it also means that there are many countermeasures. The HTTPS protocol, for example, can achieve some sort of security in the face of an attacker trying to inject false DNS information.

Overlay networks in general can be vulnerable to Eclipse attacks. In the recent studies on Eclipse attacks [85, 86, 87, 88] it has been shown that more constraints and structure changes are suggested to effectively prevent Eclipse attacks. Blockchain, however, is an unstructured network, which means that each node is connected to an arbitrary subset of nodes determined by a randomized algorithm. Many researches [89, 90] are focused on designing a new unstructured network scheme that is tolerant of these attacks.

Jesi et al. [91] proposed a new detection mechanism that can identify and blacklist

the adversary nodes. The authors state that their method is effective when a large number of malicious nodes are present in the network. This defense mechanism however does not fully preserve the open and decentralized structure of the Bitcoin network. Heilman et al. [76] study eclipse attack on Bitcoin network exclusively and present different countermeasures. These countermeasures are usually caused by the Botnet and are designed to preserve the Bitcoin's network architecture. The authors suggest techniques that can be used to save the IP addresses of the trusted nodes. If the user is connected to unknown peers, these unknown peers IP addresses are saved in a different variable namely "tried". The communication between the user and other peers in the network depends on the trust level that can be changed from time to time. The authors also mention that the attack has some specific features that make it identifiable such as a sudden TCP connection from a variety of IP addresses that send messages containing "trash" addresses. Once detected, these malicious addresses will be blacklisted from the network.

Apostolaki et al. [77] present both short-term and long-term countermeasures against Routing attacks.

The short-term measures are compatible with current protocols so the early adopters can immediately take the advantage of increased protection. These countermeasures include: selecting specific nodes with consideration of the routing, increasing the diversity of connections between nodes, monitoring round-trip time, monitoring additional statistics, embracing churn, using gateways in different AS's and preferring users hosted in the same AS and in other, commonly used prefixes. The long-term measures on the other hand, require some adjustments to the Bitcoin protocol including: encrypting Bitcoin communication and/or adopting MAC, using distinct control and data channels, using UDP heartbeats and finally requesting a block on multiple connections. Saad et al. [92] who study four forms of partitioning attacks, presented many countermeasures to these attacks.

Nayak states that stubborn mining and eclipse attacks can usually be detected by monitoring stale block rate. Stale blocks are the blocks that include a valid proof-of-work and transaction data but they are not included in the main chain. [78] The appearance of stale blocks is probable on a random basis even in absence of malicious miners. The difference is, if miner nodes are honest, the rate in which a stale block appears is the same for all of the miners. If there is a stubborn miner, the rate in which a stale block appears would change depending on the dishonest miner's strategy. An attack can be detected if by counting the number of stale blocks and comparing it with the baseline

DDoS Attacks

Countermeasures
Hashcash - A defense against DDoS attacks which later becomes the foundation of Bitcoin's proof-of-work system[93]
A static game model designed to **calculate the Nash equilibrium** that brings the best strategy for the defender[94]
Detection algorithm that relies on the **frequency sorted distributions** and the **entropy** of selected packet attributes[95]
Detection by incorporating **multiple methods of unsupervised data mining**[96]

Table 2.5: DDoS Attack Countermeasures

DDoS, or Distributed Denial of Service is a very common attack that can be used against almost any online service [97]. DDoS in general is an adversarial attempt conducted by an attacker against a target user or a network service for the purpose

of disrupting it in such a way so that the victim cannot properly operate. The attacker overwhelms the target's resources by causing a traffic jam in the network [98]. According to [43], the mining pools are the second most frequent target of DDoS attacks. DDoS attacks are very versatile and it shows that they can be executed in various ways, changes based on the different environment for it to be executed, the type of network and the nodes in the network. After analyzing different Blockchain networks, it is clear that the 51% majority attack can lead to DDoS. This is because this attack means that there is an adversary with more computation power than all other nodes on the network, so this adversary can prevent miners from adding their mined block to the network, and they can alter future transactions which can lead to denial of service.

Competing mining pools, for example, can also conduct DDoS attacks against one another. There are two objectives for conducting such an attack. First, the operations or the communications between the members of the target mining pool are disrupted or delayed. This can directly bring an advantage to the attacker mining pool in the competition. Second, as the result of the interruptions caused by the attack, the members of the victim mining pools might face frustrations and this could be a motivation for them to leave the pool.

Vasek et al. [43] states that over 60% of large mining pools have been the target of DDoS attacks. The number is reduced to 17% for the small mining pools. This suggests that the larger mining pools are more likely to be the target of DDoS attack.

In a different research [45], DDoS attacks against the mining pools were studied using Game-Theoretic analysis. The authors confirmed that there is more incentive to attack a larger mining pool than to attack a smaller one. Also the larger mining pools have more incentive to attack than the smaller mining pools since the competitors for the larger mining pools are relatively fewer and the attacker may seek to eliminate its competitors in order to increase their fraction of computational power.

There are different ways to execute this attack due to the limited number of TPB, or transactions-per-block that the Blockchain network can process in a given time. This number changes between networks, but overall there is a cap to the number of transactions per minute. According to [99] it takes on average 10 minutes to mine a new block on the Bitcoin network, with a maximum of 1 MegaByte, or one million bytes. A transaction on the Bitcoin network weighs about 500 bytes, meaning there is an average of about 2000 transactions on a single block [100]. When analyzing the amount of time it takes to mine a new block on the Bitcoin network, it takes an average of 10 minutes. 2000 transactions per 10 minutes is 200 transactions per minute. A single transaction is usually made between two different nodes on the network, so the total number of active nodes that can be served by the network per minute cannot exceed 200.

An adversary can take advantage of this and use many different nodes under his control to schedule a very small transaction (transferring 0.0001 BC for example) at the exact same time. By doing so, large numbers of users who try to make transactions at the same time will not be able to do so, causing denial of service, assuming this adversary has enough nodes to cap the capability of the network to process any new transactions during this time.

Another form of DDoS attack is executed in the memory pools, or mempool of the Blockchain network to increase the mining fee. Mempools is the place where the data that needs to be executed is stored. All of the recent and pending transactions, including unconfirmed transactions, are stored. This mempool is also a way for the network to determine the mining fee is calculated, based on the number of transactions that are in the mempool.

Saad et al. [101], introduces a new type of DDoS attack on the Blockchain network in which an adversary can create many unconfirmed transactions to significantly increase the number of transactions in the mempool. These transactions can be

either very small, or transactions that are not valid. These transactions increase the total number of transactions in the mempool and so it increases the mining fee for all the nodes in the network. This can cause users to drop and stop their mine, or create an increasing number of people who are willing to pay mining fees hoping to increase their chances.

Detection and Countermeasure: In general all large networks could be the subject of DDoS attack and before the introduction of the Blockchain based networks, many researchers had studied possible solutions to combat DDoS attacks. In [93] the author proposed a mechanism namely, Hashcash as a defense against DDoS attacks which later became the foundation of Bitcoin's proof-of-work system. Bedi et al. [94] proposed a scheme in which the interaction between the attacker and the defender is demonstrated. Then a static game model is designed to calculate the Nash equilibrium that brings the best strategy for the defender. The author examines the effectiveness of this game theoretic defense through number simulations. Feinstein et al. [95] present a detection algorithm that relies on the calculation of entropy and frequency-sorted distributions of selected packet attributes. Hyvärinen et al. [96] introduced a DDoS attack detection system that incorporates multiple methods of unsupervised data mining.

Johnson et al. [45] suggested a game theory model, to fight the different DDoS attacks that are executed on mining. This game theory model lowers the incentive of adversaries for such actions, and proves to be useful on many Blockchain networks.

Pool Hopping Attack

This attack is a very basic one where due to the increased availability of different mining pools, a single or a group of miners have the opportunity to switch between mining pools if they realize they can increase their profit by doing so. The way mining pool works is that each miner can submit his or her share of the work to solve the

Countermeasures
Detect pool hopping based on rewarding transactions - pool hoppers rewards are significantly more than that of static miners[102]
Optimized rewarding system - rewards are given to the miners in proportion to the score they have received in each round[103]
Protocol in which the **mining process is completely decentralized**[104]
Smart contract-based pool hopping prevention by sharing miners computational power[105]

Table 2.6: Pool Hopping Countermeasures

hash, so that whenever a new block is mined, the mining pool can share the reward between the different miners based on their effort in solving it. If, for example, a miner did more than half of the work, he or she will get most of the reward. However, this can cause a problem where the value of blocks that are solved in a short time are worth more than blocks that are solved in average time or more.

Lewenberg et al. [39] studied the reward sharing mechanism in mining pools by developing game theoretic models. They concluded that when the rate of transactions is high, it can be very difficult or impossible to keep the distribution of the accumulated revenue stable. Therefore, there is always an incentive for some miners to switch between pools. This pool switching can potentially sabotage the targeted mining pools if it is practiced extensively. [48] The malicious miners constantly leave and rejoin mining pools based on the expected financial rewards the pools offer.
When a mining pool offers low rewards, the attacker leaves and when the rewards are high, the attacker rejoins. The result of this leaving and rejoining the victim

mining pool by the attacker, enables him to receive more rewards than his expected hash power. On the other side, the honest miners who mine for the same mining pool consistently, receive less rewards than their expected hash power. This situation prevents the victim mining pool from operating effectively and mine blocks successfully before their competitors. [105] As a result, the honest miners will lose money by staying in the targeted mining pool and eventually have to leave the pool. Rosenfeld demonstrates in [48] that the current reward system that is practiced by mining pools, encourages pool hopping attack in the sense that it is more profitable than continuously mining.

Detection and Countermeasure: Belotti et al. [102] studied pool hopping detection exclusively. The authors propose a solution to detect pool hopping based on rewarding transactions ordering.

To determine whether a miner is practicing a hopping strategy, it is crucial to focus on his rewarding transactions. This detection strategy is based on time epoch which refers to a time window in which the miner has received rewards from different mining pools. Time epochs can be determined by analyzing the bitcoin transactions in the miner's wallet. The authors show that pool hoppers' rewards are significantly more than the static miners. Also the behavior of pool hoppers does not necessarily correlate with the value of the cryptocurrency.

Slush pool is the first mining pool that has implemented an optimized rewarding system for the purpose of pool hopping prevention. In this mechanism, rewards are given to the miners in proportion to the score they have received in each round. The scoring algorithm dynamically computes a score for each share based on its submission time. Rosenfeld et al. paper discusses a scoring mechanism that is used to compute rewards for the members of a mining pool. The proposed algorithm is based on the scoring mechanism similar to Slush's implemented method but different in a way that the scores for each share remains the same whereas.

Salimitari et al. [106] propose a prospect theory which can help a new miner to find the most profitable pool. The authors state that the best pool for a specific miner is not necessarily the best pool for all miners. The suggested utility function can be used to calculate the value function that determines the risk and loss for each miner according to their hash power and their electricity costs. The main priority to this method is loss avoiding rather than profit making. The authors evaluate the accuracy of their theoretical methodology by joining five different pools and mining for 40 days, then comparing the actual results with the predictions that are provided by the utility function. The results however show that the predictions are not as accurate as expected.

Luu et al. [104] propose a new protocol for an efficient decentralized mining pool namely SmartPool. This new protocol can be implemented in existing cryptocurrencies. It is aimed to resolve the problem with centralized mining pools in Bitcoin and Ethereum by structuring a platform where mining is completely decentralized. The authors conduct an experiment in Ethereum testnet and conclude that the protocol is efficient in practice and therefore, ready to be deployed in real blockchain networks.

Singh et al. [105] introduce a new prevention scheme for pool hopping attack prevention based on smart contracts. The main purpose of this scheme is to secure the relationship between miners by having them all to share their computational power faithfully, meaning they have to show some proof on how fast they can compute a simpler problem. This model provides number of benefits including:

1. The pool manager is able to monitor the action of each miner before they can join the mining pool

2. It requires the miners to submit coins as escrow. If a miner try to abandon the mining pool, his escrow coins are seized as punishment

3. To facilitate the calculation of the exact amount of escrow, a detailed numerical

model is provided

To examine the practicality of the proposed model, the authors implemented a case study of an Ethereum based blockchain for an IoT smart home. The authors state that only a few limitations in this model need to be addressed in future research.

More Countermeasures

Reputation-Based Countermeasures

Nojoumian et al. [1] propose a new reputation-based scheme for the proof-of-work computation. The authors rely on game theoretical analyses to demonstrate the effectiveness of their proposed model. This solution is meant to encourage the miners to stay away from dishonest mining strategies by relying on a system of reputation. Each miner is assigned a public reputation value and this value reflects how trustworthy the miner has been so far in terms of mining activities. The authors demonstrate that, by using this solution, honest mining becomes Nash Equilibrium. It means that the miners who are more reputable, have a higher chance of receiving mining invitations from the pool managers than the ones who are not trustworthy. As a result, the miners are incentivized to maintain their reputation continuously. Even if miners can increase their short term revenue by mining dishonestly, in the long term, it will be in their best interest to maintain their reputation value high.

Tang et al. [107] also propose a reputation-based approach to block the miners' malicious activities. With this approach, the satisfaction of the pool managers towards each miner is recorded and evolved overtime. Before beginning the mining process, the pool manager sets a threshold level for reputation and the miners whose reputation value is below the desired value, are not eligible to join the pool. In this model the reputation value is also used to judge whether the peer is reliable or not. The authors hope that by implementing this solution, the pool managers only select trustworthy miners and overtime the development of the cryptocurrency system

benefits from this reputation-based scheme.

Application Countermeasures

Using any cryptocurrency requires the use of an application. Based on the nature of applications, they will have their own advantages and disadvantages when it comes to security.

There are various possible application countermeasures that are used to block potential adversaries. For example, users can create a backup of their cryptocurrency wallet so that they can have back access to the hash in case someone manages to break into it, users can also change their password periodically to ensure wallet encryption. There are plenty of available services online and offline to increase the application security.

2.5.2 Technical Discussion

Blockchain technology has become more and more popular in recent years. This popularity also means that more people are trying to find a way to take advantage of this technology. The mining attacks that were reviewed in this dissertation are some of the most common recent ones. Most of the attacks and countermeasures that were presented in this dissertation are relatively new, which shows that as this technology becomes more and more accessible, so is the number of attacks increases.

DDoS was known to be the most common mining attack in the Blockchain network, however in the last couple of years the 51% attack took its place as the most used attack against miners on the network. With cases such as the Bitcoin Cash attack during May 2019, where two pools carried a 51% attack on the BTC Blockchain in order to stop a miner from taking coins, Or the Ethereum Classic January 2019 attack.

Blockchain is a growing technology that is used in many applications such as smart contracts [108], IoT [109] and most commonly, cryptocurrencies [110].

Blockchain applications provides a decentralized exchange environment which is built on cryptography and peer-to-peer technologies. These technologies make it so that accepted "transactions" are written on an immutable ledger [111, 112]. This ledger is public, and its peer-to-peer system means that it is maintained in a decentralized fashion.

Bitcoin [22], for example, is a cryptocurrency that has gained a substantial popularity since its introduction in 2009. Unlike the traditional currency system that requires an intermediary to keep track of the transactions, Bitcoin transactions are recorded as ledgers and distributed among all the participants on the network, therefore, all the transactions are public. This public log is known as Blockchain. All peers in the Bitcoin network have full access to the history of transactions and due to the cryptographic relation of each block to its previous block, it would be extremely difficult for an adversary to modify a block once it is broadcasted to the other participants in the network.

A new block can be added to the Blockchain in a process called mining. Mining refers to finding a 64 digit hex hash value, namely "nonce", that must be less than or equal to the target hash. Finding this cryptographic hash value can be done in several ways, depending on the Blockchain network. Most common one is the Bitcoin Blockchain that uses Proof-of-Work (PoW) where a miner must solve a challenging mathematical problem in order to acquire the hash value for the next block. This process requires intensive computational work. Once the correct value is found by a miner, it can be easily verified by other miners in the network. A miner who successfully carried out the PoW obtains bitcoins as a reward for his work.

Due to the extremely competitive nature of mining, a miner whose computational power is only a small fraction of the whole network's computational power, has a very low chance to mine a block. Therefore, miners often join mining pools to increase their revenue. Once a mining pool generates a new block, the obtained revenue is

distributed among all pool members with the respect to their computational power. The availability of numerous mining pools, brings the opportunity to a miner to switch between pools if he realizes that the new pool can potentially increase his profit. Lewenberg et al. [39] studied the reward sharing mechanism in mining pools by developing game theoretic models. They concluded that specifically when the rate of transactions is high, it can be very difficult or impossible to keep the distribution of the accumulated revenue stable. Therefore, there is always an incentive for some miners to switch between pools. Mining a new block can also be done using the Proof-of-Stake (PoS) method [24]. This is an alternative to the PoW, which requires a huge amount of energy.

2.5.3 Concluding Remarks

The Blockchain-based cryptocurrencies have gained substantial attention since the introduction of Bitcoin as an alternative decentralized currency. However, this popularity has also attracted individuals with malicious intentions. The usability of Bitcoin and other similar cryptocurrencies depends on the level of their reliability and security. Mining attacks have been one of the main issues of the Blockchain-based cryptocurrencies. In this dissertation, we described these attacks individually and then reviewed the detection methods and countermeasures that have been proposed by many researchers. Most of these solutions target a particular kind of attack. In contrast, the reputation-based solutions do not address a specific mining attack. Instead, they are aimed to encourage miners to commit to honest mining strategies. Many of these solutions are validated by relying only on game theoretic analyses whereas many other solutions are validated by incorporating practical experiments. To clarify the effectiveness and the practicality of these solutions, further research studies with real world experiments are needed.

CHAPTER 3

ISRAFT - DECENTRALIZED INFORMATION SHARING

3.1 INTRODUCTION

With the development of information sharing technology and its increasing threats, various secure information sharing techniques have been developed over years. One challenge is to design a closed network system where information can be shared in a secured way under the assumption that data is tamper-proof and secured in the presence of adversaries. Raft consensus algorithm, designed by Ongaro and Ousterhout [113], seems to be one of the initial promising solutions to this problem. The original Raft implementation is an improved Paxos style protocol that is much easier to understand. It offers the same security features as Paxos. The algorithm was designed in the context of replicated state machines in which the algorithm keeps the logs consistent and identical even when a subset of servers are down. This algorithm performs well considering its main objective, i.e., handling servers communications with a client and synchronizing their data. However, to be used for information sharing, it needs to be improved.

The concept of achieving consensus among parties has been studied for many years. It became well-known when Satoshi Nakamoto published the famous Bitcoin paper in 2009 [114] and introduced Blockchain as an infrastructure for the Bitcoin technology; see [1] for details of Bitcoin mining in Blockchain. Achieving consensus among peers on the network was necessary for implementation of Bitcoin. This platform was the first state-of-the art that utilized a consensus algorithm to achieve trust among parties in a decentralized network. Satoshi's idea was based on the one-cpu-one-

vote rule, i.e., proof-of-work (PoW) consensus algorithm that has been and still is the most commonly used consensus algorithm for cryptocurrency. Other consensus algorithms were developed ever since, such as proof-of-stake (PoS) [24] and proof-of-authority (PoA) [115], to name a few. Paxos is an example of a fault-tolerant distributed system that is complex to be implemented. However, Raft was introduced as a modern consensus algorithm that is based on the same principles of Paxos but with less complexity.

Drawbacks of Information Sharing Using the Original Raft.

We define *information sharing* as the possibility of servers to send secure messages to a central log via a secured and tamper-proof channel in the presence of adversaries. The original Raft algorithm is the first building block to be used for this purpose. However, it has many disadvantages. The first and most important is that Raft can not operate properly when an adversarial node is presented in the cluster. Here, we illustrate a couple of issues when the original Raft is used.

Leader Election: The first stage of the system is to elect a cluster leader. In a malicious environment, a server can self-elect by updating to a new term and sending heartbeat messages to the other servers in the cluster. A malicious leader can also stall the system by initiating election repeatedly, thus preventing the system's availability, i.e., causing a stall election. The servers can also start a new election even if it still receives heartbeat messages from the current leader.

Log Replication: This happens when a new state is presented to the system and the servers eventually are required to arrive at the same state. Replicating the log needs to be secured and immutable as possible, especially when an adversarial server is present. In the Raft algorithm, the leader server can change or drop requests that are sent to him, i.e., violating the integrity of the system.

Another drawback is that the Raft algorithm is a client-server communication with

a single client and a cluster of servers, whereas we need the servers to send requests to the cluster without relying on a single authority. This leads to a multi-client-server communication where each node is both a client and a server.

Improved Raft for Information Sharing.

We present an alternative method of information sharing in a closed and isolated network with N servers. The communication model is similar to that of the original Raft under the assumption that it is reliable in the presence of an adversary that can cause network lagging and information loss. Therefore, we incorporate the following improvements into the Raft information sharing design:

1. *Byzantine Fault Tolerance:* As stated earlier, it is impractical to use the original Raft as the main consensus algorithm without modifications. The first thing that needs to be taken into account is to allow the system to work under the Byzantine Fault Tolerance (BFT) assumption [116]. This assumption allows the system to work securely in the presence of Byzantine servers that can sabotage the Raft algorithm. Copeland and Zhong [117] provide *Tangaroa* as a solution to this problem.

2. *Client servers:* The Raft algorithm is built as a cluster of servers that communicate to a single client. Whenever the client wishes to update the log, he sends a message to the leader server that handles the log replication for all other servers in the cluster. Non-leader servers cannot send any request to change the data on the log. In our design, every server is able to request a data appendance. This is similar to the state-change in the state machines.

3. *Log modification:* In the original Raft, every server holds a log, where every server keeps the machine states. In our implementation, the log needs to be changed into a data buffer that holds data. The simple state log then turns into

an immutable data log. A server can only append information to this log to be shared with other servers in the cluster. For the sake of simplicity, *data log* is used when we refer to the buffer.

4. *Validation:* Data validation can be subjective to the purpose of the network. We provide a design in which every server can share data in the cluster regardless of the meaning of data. The validation process can be added as a second layer to the infrastructure so that servers can be dropped from the cluster when they don't follow the required messages properties.

3.2 ISRAFT: OUR NEW INFORMATION SHARING PARADIGM

Our improved version of the Raft consensus algorithm, named *ISRaft: Information Sharing Raft*, uses similar fundamentals as the original Raft algorithm, i.e., Leader Election and Data Log Replication. In addition to modifying these features, our new design includes message validation and security in the presence of an adversary. This design of Raft offers the same security properties as the original Raft consensus algorithm that are as follows:

- *Election Safety:* Only a single leader is allowed per cluster.
- *Leader Append-Only:* The elected leader can only append information to the data log. A server can never delete or overwrite any entry.
- *Log Matching:* If there are two data logs on two different servers with the same term and same data, then the entire data logs on the two servers are identical in all entries up to the last term.
- *Leader Completeness:* When a leader commits a message to the log, that message will be present in all future logs of the leader.

There is one more property that the original Raft consensus illustrates, which is the *State-Machine Safety*; however, due to the nature of our design, it is not essential to maintain this safety property as explained later. Compared to the original Raft algorithm, the major change for our design is the ability to have the servers communicate independently and also have them send data to the leader to request for an update of the data logs. This means the *remote procedure call* (RPC) messages that are used in the original Raft as well as the protocol itself must be modified so that they fit into our design.

First, we explain the RPC requests that are used in the modified algorithm. We then review every stage of the information-sharing process to make the whole procedure clear. The Raft algorithm proposed the following two RPCs: *RequestVote* and *AppendEntries*. However, the ISRaft utilizes a modified adaptation and a couple of other RPCs as follows:

- *RequestVote:* This is initiated by candidates in the cluster. This RPC will be sent anytime a server hasn't received any heartbeat message from a leader.

- *AppendEntries:* This is initiated by the leader of the cluster. This RPC is sent as a request to replicate the log on all servers in the cluster. This message is also used as a heartbeat when no log entries are attached to the RPC.

- *AppendEntries-Response:* This is a response that is initiated by any server receiving *AppendEntries* RPC. The RPC contains the data to be appended as well as the signature of the server.

- *RequestAppend:* This is initiated by any server in the cluster to add information to the log.

3.2.1 Architecture of the ISRaft

In our setting, a server in the cluster can be in one of the three states: *leader, follower,* or *candidate*. A leader is the only authority that can append entries to the log. The other servers can only request the leader to append messages. This prevents the overflowing and message duplication problems. Servers on the cluster have the certified public-key of all other servers. Therefore, when a server sends any RPC request message, he signs it with his own private-key. Servers reject any RPC message that does not include a valid signature. Any *public key cryptosystem* (PKC), such as RSA, can be used in our model. When a leader appends a message from the *RequestAppend* RPC, he adds the public-key of the requesting server to the data log. This way, every data log entry can be traced to the server that has initiated the request.

Similar to the original Raft, our ISRaft algorithm uses terms. These terms are numbered with consecutive integers starting from 1. It is incremented every time a new term is declared by the leader. Every term begins with an *election*, i.e., a process by which the candidate servers send a RequestVote RPC to the cluster. When a candidate receives the majority of votes in the cluster, e.g., $n/2 + 1$ for a cluster of size n, he declares himself as the new leader and then he increments the term number. The leader can then append the new message to the data log and share it with the servers in the cluster. Any new entry in the data log contains the hash of the previous entry, which can be computed by all servers. To compute a hash value at index i, the server computes the hash of the data at index $i - 1$. When two servers agree on a hash value for index i, they verify to make sure they have identical log entries up to index i.

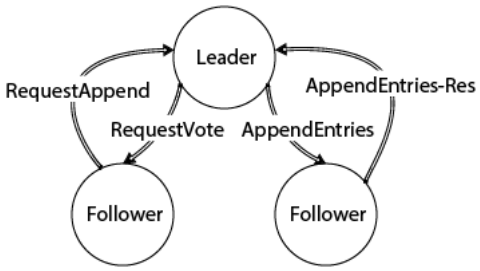


Figure 3.1: PRCs that are used in this model.

3.2.2 Leader Election

The first step of the algorithm is to elect a leader server from all servers in the cluster. All servers start their life as a follower, a state in which the server awaits a heartbeat message from a leader in the cluster. If no message was sent after a predefined period of time, called *election timeout*, the follower changes its state to a candidate. It then begins the election process. The candidate server increments the current term and he sends *RequestVote* RPC concatenate with its own signature to all servers in the cluster. A follower or a candidate server that receives the RPC with a valid signature will send a vote if and only if the following conditions are met:

1. The server is a follower or a candidate.
2. The server has not received any heartbeat message from the current leader.
3. The new term is, at least, the current server's term plus 1.
4. The *RequestVote* is signed with the candidate private-key.

A server that receives the first *RequestVote* RPC for any term will hold the first vote until the end of the *election timeout*. If no other *RequestVote* RPC arrives, it will vote for the first candidate. However, if a second *RequestVote* RPC arrives during that time from a different server, it will immediately vote for the second server. This is to prevent a case where a server starves the system by initiating election constantly.

When a server replies with a vote, it will update its term number and change its state to a follower. The server will then wait for the first heartbeat message from the elected leader, and if none arrives at the expected time, it will become a candidate himself. A candidate wins the election if he received the majority votes from the cluster. The elected leader then changes its state to a leader and sends a heartbeat *AppendEntries* message that includes the signed vote messages that he received from the servers in the cluster. This will be a proof of election and also prevents a self-promoted leader.

While waiting for votes, a candidate may receive an *AppendEntries* RPC from another server claiming to be the leader. If the term in the RPC is at least as large as the candidate's current term and the RPC message contains the signed votes of the majority servers, the candidate recognizes that server as the actual leader and returns to the follower state. Otherwise, the candidate rejects the RPC and continues to remain in the candidate state.

3.2.3 Log Replication

In our setting, each follower can request an update to the data log from the leader via the *RequestAppend* RPC. This RPC contains the signature of the requesting server and the data itself. The signature guarantees the authenticity of this request, which prevents the malicious servers from forging other server's requests. The server sends the RPC to all other servers in the cluster including the leader. This is to prevent manipulation of data and also to guarantee the leader receives the message in the case a new leader has been elected without the followers' knowledge. The leader then validates the message's signature and begins the data appendance to the data log.

To start, the leader updates its own data log with the new data and increments the term. It will then send *AppendEntries* RPC to all servers in the cluster. This message contains the new data log entry to be appended and all signed votes from

the majority of the servers in the cluster. When a server receives the *AppendEntries* RPC, he performs the following tasks:

1. Compares the log entries received from the leader and the requesting server. If the leader itself has requested the update, he skips this procedure.
2. Verifies if the leader's term is at least equal to the requesting server's term.
3. Validates if the leader's votes are legitimate.
4. Verifies if the hash of the RPC message is the expected hash value.

If all these conditions are satisfied, the server appends the message to the log. Otherwise, the server will reject this message, and if no other *AppendEntries* message arrives in the chosen period, the server becomes a candidate. When a server first receives the message from the leader, he will respond with the *AppendEntries* response to all servers in the cluster. A server commits the appended data log only when he receives a similar *AppendEntries* response from the majority of the servers for the same term and data. There might be a case where a server sends the *RequestAppend* RPC to the cluster, including the leader with the data to be appended, but the leader sends the *AppendEntries* RPC without the appended data. This can happen if the leader never receives the RPC from the server, or if the leader is malicious. In both cases, the servers in the cluster will not be able to commit the data log, and as a result, the requesting server will keep sending the *RequestAppend*.

3.2.4 Validation

Servers constantly validate their data log with every heartbeat message from the leader. The hash value of the latest data log needs to be in correlation with the server's last data entry. It's also necessary to validate the integrity of the message. This can be done by analyzing the data log itself. Our model doesn't restrict any type of data

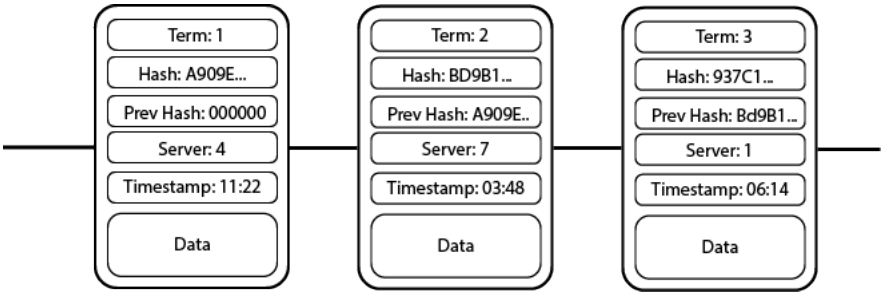


Figure 3.2: Data log example.

to be appended to the log, therefore, servers can send any unrestricted data, i.e., our setting doesn't address this type of validation. However, future implementations can have special validation mechanisms to define specific restrictions on the data to be appended.

As an example of validation, we can refer to a cluster that manages relatively complex equations that can be verified by servers. A server can send an equation along with its solution to a cluster. Subsequently, if the majority of parties verify this solution as a correct one, the server then commits it to the data log.

3.3 IMPLEMENTATION

We implemented the proposed model with C, as a low-level programming language, so that it can be executed on resource-constrained devices that have limited memory and computational power. This implementation also prevents overheads that might be imposed by higher-level programming languages.

The initial program was executed on a Linux environment and then ran on 10 Raspberry Pi's Model 3, all with similar properties, i.e., same clock speed and memory usage. Note that we had to use a programming language that supports the unusual communication properties of RPC when compared with TCP/IP or other protocols. The later model was then implemented on AWS servers, where we had the capability to turn on and off any number of small servers as needed. Some servers in the system

were then given the 'malicious' flag, which causes it to act in a non-deterministic way, e.g., spreading false messages, not replying to votes, and clogging the network. In an alternative implementation, we incorporated partitions into the cluster in order to measure and analyze the time it takes for a cluster to elect a new leader and operate when a partition is placed and/or removed. Our implementation is available at: `www.github.com/Morters/Raft_C`.

3.3.1 Implementation Architecture

As stated earlier, our ISRaft design was developed by C with IPC and RPC libraries. After establishing the commands, it was tested with CuTest on a Linux environment to evaluate its operation. The code is then compiled and ran on a few Raspberry Pi 3-s, and later on multiple Amazon AWS EC2 servers, each operating as a node and communicating with other EC2 servers on a RPC protocol over socket communication. The EC2 servers that we used are t2.nano with 512 MiB of memory and 1 vCPU running at 3.3 GHz. We implemented the RPCs as functions where each RPC can be called by every node on the system. The functions and protocols are organized in the header file *CRaft.h*.Our major goal was to make the minimal changes needed to achieve the same security level as the original Raft. We also intended to make our implementation modular enough so that it can be executed on the majority of devices. Every server was built as a separated AWS EC2 Apache that is identified by a unique IP address and ID.

In our implementation, the servers can send data over socket communication, however, any communication protocol can work. Every server then generates an RSA private-key and public-key pair that is shared with the cluster. Every time a new node enters the cluster, it is required to send its credentials to all servers in the cluster. Accordingly, all servers are required to update their server database. Our implementation does not address a case where there is an error with the credentials

or when a server is disconnected from the network since these are out of the scope of our work - although they can be addressed by other mechanisms.

Every server has a *raft* data structure that contains necessary information regarding that specific server. The state includes: `current_state` - current state of the server; `current_term` - integer term number; `voted` - a flag that states whether the server voted for the current term or not; and `datalog` - chain of datalog entries. The datalog size is limited to the storage capability of the server. The log can be stored and accessed on a cloud secured server, however, our design uses an offline information sharing model in a closed environment.

The next phase is to test how our ISRaft addresses malicious behaviors. To validate this, we created specific nodes, called 'malicious, 'which act in a non-deterministic way as explained earlier. For an easier implementation, we used Python to inject these malicious nodes into the cluster randomly. The malicious nodes could have message droppings, datalog mismatching, message duplication, to name a few. For the partitioning, we simply restricted the communication among different servers to simulate a network crash. When needed, the partition could be lifted by reconnecting the servers. Below are our RPC pseudocodes:

3.4 ANALYSIS OF OUR ISRAFT

The analysis of our model contains two parts. We initially compare the ISRaft with similar consensus algorithms. Subsequently, we analyze the correctness and efficiency of our model, especially when it's compared with the original Raft algorithm and the similar Tangaroa Raft implementation.

3.4.1 Comparison

The comparison, shown in Table 3.3, is partially based on Bamakan et. al. [118] where several consensus algorithms were evaluated with the same criteria. The first

Algorithm 1 RequestVote RPC

Request

hashed_req_vote `/* Vote for the server initiating the RPC */`

Response

user_term ← current term number

voted ← 0 `/* Flag of the server vote */`

if candidate_term < user_term **then return** ERROR

else if voted ! = 0 and candidate_ID ! = NULL **then**

    vote ← Encrypted(candidate_votes)

end ifreturn SUCCESS

Algorithm 2 AppendEntries RPC - Request

log_entry `/* Log data to be added */`

leader_term `/* New log term */`

signed_votes `/* Cluster votes of the leader. Data is encrypted with the private-key of the servers and will be decrypted with the public-key */`

comparison aspect is the type of chain that the model will most likely be implemented with, e.g., whether the model is using blocks or log. This can be determined by a number of factors, however, for this comparison we simply take the most commonly used chain method for each of the models. Our ISRaft as well as Tangaroa are based on the original Raft algorithm, therefore, they use a similar *log-based chain* method. Paxos and BFT are based on achieving consensus among parties without having any logging method such as block/log.

The second criteria is the type of Blockchain. Generally, the *permission-less* and

Algorithm 3 AppendEntries RPC - Response

```
n /* The number of servers in the cluster */
user_term /* The latest term of the current server */
prev_log_entry /* The previous log entry */

if leader_term < user_term then return FALSE
else if sec_check ! = TRUE then
    /* Check if the signed votes are valid for at least 1/2n+1 for n
servers in the cluster */ return FALSE
else if leader.prev_log_entry ! = user.prev_log_entry then return FALSE
end if
log.append(log_entry)
if leader_term > current_term then
    current_term ← leader_term
end ifreturn SUCCESS
```

Algorithm 4 RequestAppend RPC

```
log_entry /* The entry to be appended */
send_encrypt(log_entry) /* Encrypt with private key */
```

the *permissioned* are the two main ways for using Blockchain. Permission-less is also known as public, whereas the other is private. PoW is a permission-less Blockchain where the consensus can operate on an open environment and it is proof-based. The Raft, Paxos, Tangaroa, and BFT are permissioned Blockchain meaning that they operate on a closed environment with some sort of centralization with servers-client communications. Our ISRaft is somewhere in the middle. It was designed to run on an open environment where all nodes operate with the same consensus algorithm and

new nodes can join the clusters. The election, however, is voting-based that is similar to most permissioned Blockchains.

Third comparison is the trust model. The network can be *fully-trusted* where the entire network communication is moderated and all nodes can be trusted; *semi-trusted* where part of the network can be moderated and access is granted only to trusted nodes; and finally, *un-trusted* where the network is not moderated. Our ISRaft is categorized as semi-trusted, however, it can be easily modified to be un-trusted if needed. In the un-trusted model, servers can join and leave clusters without approval as long as they provide the appropriate public-key.

The *degree of decentralization* shows how the model is decentralized in comparison to others. When a models' node is dependable on other nodes, the degree of his decentralization is weakened. The original Raft, for example, has low degree of decentralization since different commands can only be initiated by the client to the servers.Our ISRaft is different from all other permissioned algorithms due to the fact that it has a high decentralization degree as the nodes can communicate with each other and achieve a consensus without the intervention of a server node. The *scalability* is how well the model can operate on a larger scale, requiring more computational power and resources. Our ISRaft is not different from the original Raft consensus algorithm form this perspective. The last comparison is the *percentage of byzantine nodes* the model can tolerate. Our ISRaft is an upgrade to the original Raft as the algorithm can tolerate up to 50% of byzantine nodes.

Model	Chain	Type of Blockchain	Trust Model	Degree of Decentralization	Scalability	Byzantine Fault Tolerance
ISRaft	Log Based	Semi-Permissioned	Semi-Trusted	High	Moderate	50%
Raft [1]	Log Based	Permissioned	Semi-Trusted	Low	Moderate	NA
Paxos [14]	Participants	Permissioned	Semi-Trusted	Low	Low	NA
Tangaroa [13]	Log Based	Permissioned	Semi-Trusted	Low	Moderate	50%
PoW [15]	Block Based	Permission-less	Un-Trusted	Very High	High	50%
BFT [16]	Participants	Permissioned	Semi-Trusted	Low	Low	33%

Figure 3.3: Comparison of consensus algorithms.

3.4.2 Efficiency Analysis

Our implementation used EC2 Amazon servers running in a cluster. Each server is an EC2 instance that uses multi-threading as part of its infrastructure. The efficiency tested in this section is the average time it took for a cluster to achieve consensus for the same election timeout value. The comparison of the model is done only with the original Raft and the similar modification of Raft, named Tangaroa [117]. Tangaroa was implemented using similar tools as our ISRaft.

The algorithms were then tested on different similar environments: one with no malicious node active and another with 20% of malicious nodes who can operate in one of three ways: Holding messages, pretending to be different servers in the cluster, or sending random messages.Each malicious node was given a random assignment to either one of these operations.

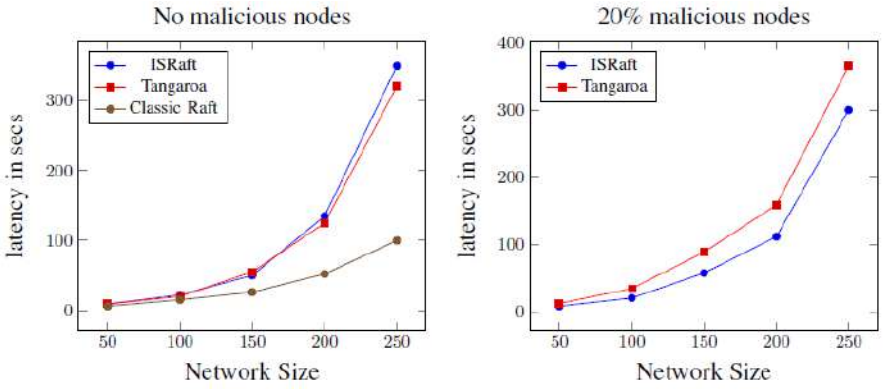


Figure 3.4: Algorithms latency with different % of malicious servers.

Figure 3.4 shows the result of our implementation with respect to the network size (number of nodes) and the total latency of the network for all communications. The latency was calculated by having each node store the time it initiated the command and have it reduced from the end time when the command reached all other nodes. The Raft operates faster than ISRaft and Tangaroa in a network with no malicious modes. This is due to the fact that the servers had to store the entire cluster's keys

and many RPCs so that they operate properly, whereas in other models, the nodes only require to store the log and term.

Malicious nodes have a significant effect on the operation of the classic Raft algorithm. The results were always inconclusive as the model kept stalling and messages were not passing through. This is why the original Raft model was not fully tested in this malicious environment. The ISRaft and Tangaroa were both capable of handling 20% of malicious nodes and were given similar results as the one with no malicious nodes.

This analysis illustrates that the original Raft is incapable of handling malicious nodes and it is much faster on a large scale than our ISRaft. However, in real-world applications where malicious nodes exist, the ISRaft operates as expected. The main difference between our ISRaft and the Tangaroa is that the Tangaroa is not as decentralized as our ISRaft and it is required to operate under server-client communication whereas our ISRaft is fully decentralized.

CHAPTER 4

CONSENSUS ALGORITHM FOR AUTONOMOUS UNITS

4.1 INTRODUCTION

Autonomous units can operate under many different operations and models. Much like a human, each unit is required to make decisions and be able to communicate and act upon it. Comprising many small individual nodes, autonomous units in decentralized fashion have no centralized authority to guide them. Instead, they rely on individual communications in order to complete tasks, achieve consensus and share information [119]. These groups are robust, flexible and scalable, which allows them to complete many different tasks in different fields. Decentralized autonomous units contain many individual nodes with several advantages. The loss of a few nodes will not affect the system, contributing to its robustness. Flexibility is possible because of simple communications that take place between nodes, and scalability works since this method of operation can work together with different numbers. Autonomous units can be applied in fields that deal with data collection and management. Some examples are exploration, surveillance, and data classification. The methodologies of such systems are efficient by themselves, but when implemented with other blockchain technologies, many new opportunities arise.

The Blockchain infrastructure was introduced in 2009 through the creation of Bitcoin. Bitcoin required the nodes of a system to come to a consensus in order to append blocks to the blockchain. In most blockchain systems, the *Proof-of-Work* (PoW) consensus protocol is used. However, this protocol is known to be slow and energy consuming, which makes it impractical for many applications. Blockchains

are being established in many fields of technology in order to improve security and provide features such as decentralization. One of these applications is in autonomous driving [120], where security is extremely important in order to establish autonomous vehicles that are reliable and safe. Blockchain technology has also extended to Robot Swarms. The presence of a blockchain in robot swarms allows for these nodes to have easier and more efficient communication. Blockchains allow robot swarms to have access to shared knowledge while still retaining the individual communications that take place in swarms. Furthermore, the immutable characteristics of the blockchain also enable robot swarms to be more secure and reliable in terms of data storing and sharing.

This dissertation therefore proposes an alternative version of the *Information Sharing Raft* (ISRaft) consensus protocol [121] to allow communication among nodes in a secured fashion while maintaining the security features of the original ISRaft algorithm even in the presence of adversarial nodes. ISRaft builds upon the Raft consensus protocol that achieves consensus through the process of leader elections, voting, block generating and validating. ISRaft differs from the original Raft by adding Byzantine Fault Tolerance, allowing all nodes in a network to request data changes, validation of any data types, and adding reputation value to each node. In ISRaft, individual nodes in the system are able to communicate with each other through *Remote Procedure Calls* (RPC).

4.2 ISRAFT CONSENSUS PROTOCOL

In this section, we present the ISRaft consensus protocol in a closed and isolated network. ISRaft protocol uses similar fundamentals as the original Raft algorithm, i.e, Leader Election and Server Communication. In addition to modifying these features, this new design includes a trust value, message validation and security features in the presence of an adversary.

4.2.1 Design Overview

The main idea of ISRaft protocol is to have all nodes in the cluster to hold an agreed upon ledger, containing messages of communication among different nodes. A node in the cluster can be in one of the three states: *follower, candidate* or *leader*. A leader is the only authority in the cluster that can append blocks to the chain, making it the miner of the current block. The other nodes can send messages to each other and the leader, and the leader then has the responsibility to distribute the message among all other nodes. For simplicity, communication among nodes in the network is done via RPC messages for minimum coding and faster initialization.

1. **Asymmetric Encryption:** Each node is initialized with a pair of cryptographic keys used for authentication and signature. The public-key is submitted and logged into a registry that holds all identities in the network. Other nodes will receive an update whenever a new node joins the network. Whenever a node sends any RPC message, he signs it with his own private-key. Nodes reject any RPC message that does not include a valid signature. Any key pair, such as RSA, can be used for the purpose of this model. This also helps with tracing back messages when a trust is broken between nodes.

2. **Permissioned Blockchain**: New nodes need to go through an enrollment process where a key pair is initiated and stored on the registry along with its address, trust value, and other required parameters.

3. **Secured Channel:** For simplicity, we assume that the communication of the nodes is done on a secured channel, without the possibility of having a man-in-the-middle attack. This, however, can be easily addressed in future works. Furthermore, communication is done via RPC messages, signed by each node.

As mentioned, a node in the system can be in either one of the following identities: *follower, candidate* or *leader*.

- **Follower:** This state is the initial state of a node upon joining the network. While in this state, the follower operates under the current leader of the cluster. If a follower node wishes to add anything to the chain, the leader needs to first validate the message, and only then it can be committed. A follower receives a stream of heartbeat messages from the leader to make sure it is updated to the last block. A heartbeat message is sent every short period of time, called *election timeout*, and it can include new transactions, data and/or leader change.

- **Candidate:** When a follower does not receive heartbeat messages within some predetermined period of time, it automatically changes its state to a candidate. While in this state, the node sends a request-vote message to all other nodes in the cluster, asking for their vote so that he can become the leader, and so the miner of the next block. A candidate can not vote for himself.

- **Leader:** A candidate can become a leader when he receives at least $n/2 + 1$ votes in a cluster of n nodes. A leader has the responsibility to mine the next block, and to send heartbeat messages to all nodes in the network.

Trust between two nodes plays an important factor in this ISRaft protocol. Trust represents the confidence in which nodes rely on each other. Let P_i be a node with a public-key identifier i. P_i then holds a trust value associated with each node in the network. The trust value assigned by P_j to P_i is then T_i^j. This value is in the interval $[0, 1]$. These values are unique and do not have to be symmetric, i.e., $T_i^j \neq T_j^i$ [122]. When a node sends an RPC message of any kind (excluding heartbeat message and responds) to the other nodes in the network, the recipients immediately decrease the trust value of the sender by a fixed small amount. This is done as a way to reduce the total number of messages that can be sent every single second, decreasing the possibility of overflowing and DDoS attacks on the network by any node in the system.

ISRaft uses the following RPCs for its operation:

- *RequestVote* - This RPC is sent whenever a *candidate* is calling for votes.
- *RequestAdd* - Every node in the cluster can send this message to the leader, asking for some data to be added to the next generated block.
- *AppendBlock* - The leader can send this message whenever a new block is mined. This can happen every fixed time, or by the decision of the leader.
- *ApproveCommit* - After a new block has been added to the node, he sends this RPC back to the leader.

These simple RPCs are what makes this consensus protocol easy to implement and operate in resource-constrained devices.

4.2.2 Leader Election

For this protocol to work, a leader must be elected. This process is done automatically when no heartbeat message is being sent over *election timeout*. After that period of time, the *follower* node changes its state to a *candidate* and he begins the election process. During this time, the node sends *RequestVote* RPC concatenate with the latest block number signed with his private-key signature to all nodes in the cluster. A follower or a candidate node that receives the RPC will then send a vote if and only if the following conditions are met:

- The node has not received any heartbeat messages from the current leader.
- The latest block number is at least equal to the current node's term plus 1.
- The *RequestVote* RPC is signed with a valid candidate's private-key.
- The trust value T of the sender is at least 0.5.

A node that receives the first *RequestVote* RPC will hold the first vote until the end of the *election timeout* regardless of the candidate's trust value, as long as the candidate fulfills the conditions. However, in the case where more than one *RequestVote* arrives, the node will choose who to vote for based on the trust value. A candidate with a higher trust value T will have a higher chance of getting the vote. A leader is elected when he receives the votes of the majority of the cluster. At that point, the elected leader sends out an RPC that includes the signed vote messages that he received from the nodes in the cluster. This acts as a proof of election and also prevents a self-promoted leader. All other nodes in the cluster then increase the trust value of the appointed leader by some fixed amount. This can be seen as a reward given to the elected leader who will then mine the next block.

After the new leader has been elected, he starts sending heartbeat messages containing the signatures of all other nodes who voted for him as a continuous proof of his leadership. If a new node was not aware that a new leader had been elected, the heartbeat message serves as an update for the node to follow. A leader is elected for a limited amount of time, called *leadership term*. At the end of this term, the leader gets to mine a new single block added to the chain and then a new election process begins.

During the election process, a node can receive a heartbeat RPC from a different node claiming to be the new leader. If the block number of the new heartbeat is at least higher than the block number of the previous heartbeat message, the node will recognize that new node as the leader and will follow the new leader. Otherwise, it will reject the heartbeat RPC and will continue with the election process.

4.2.3 Block Generation

Every leadership term ends with a block being appended to the chain. A newly created block has the following data, as shown in Figure 4.1: block number, block

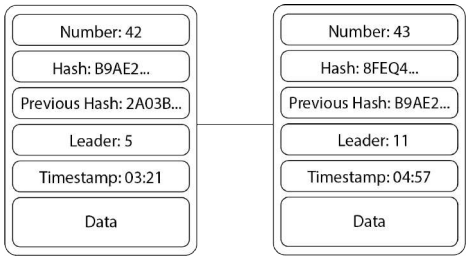


Figure 4.1: Example of ISRaft blocks.

hash, previous hash, block leader, timestamp, and data array.

When a follower node intends to add data to the block, he sends a request via the *RequestAdd* RPC. This RPC is signed by the requesting node. The signature guarantees the authenticity of the request, which prevents a malicious node from forging another node's request. This RPC is sent to all nodes in the cluster including the leader. This is performed for two purposes: To prevent manipulation of data, and to guarantee that a leader will receive the data in the case if a new leader has been elected without the node's knowledge. When the *leadership term* is coming to end, the leader begins the new block generation process by hashing the value of the new block and sending the *AppendBlock* RPC to all nodes in the cluster. This message contains the hash value of the block to be appended and all signed votes from the majority of the nodes in the cluster. When a node receives the *AppendBlock* RPC, he performs the following:

1. Compares the hash in the RPC message to make sure it is the expected hash value,

2. Verifies if the block number is larger than the last committed block, and

3. Validates if the leader's votes are legitimate.

If all conditions are satisfied, the node increases the trust value of the leader and sends a signed *ApproveCommit* RPC to the leader. If the majority of nodes in the network send this approval message, the leader can then send a second RPC that

includes the signed approvals of all nodes in the network. When a node receives the second RPC, it commits the new block to the chain.

4.2.4 Data Update and Validation

Nodes are able to add data to the block through a request made to the leader of the current term. The leader then decides to store the information received from the nodes into the next block. The data is locally stored in a 2D array on each node, where the first dimension is the number of nodes in the network and the second dimension is the decision made by each one. The decision of each node will be visible to others and will be used to modify the trust values of other nodes.

Every task is bound in time, and during that time, it is expected that the system will achieve a consensus. Examples for such tasks can be learning the color of the surface, reading texts, or identifying threats. When a node makes a decision and commits it to the blockchain, other nodes can read that decision and change the trust value of that node accordingly. For example, if the system needs to identify the color of the surface, each one of the nodes will read the color and submit its decision to the current leader, who will in turn commit these decisions into the next block. At the end of the task, the last block can be read to make a final decision and validate it. The validation process heavily relies on the trust values of nodes in the network. When a consensus is required, the reputation value of each node is calculated based on individual trust values, which is defined as follows [123]:

Definition 4 *Let T_i^j be the trust value assigned by P_j to P_i. Let T_i be the reputation function that illustrates how trustworthy P_i is:*

$$T_i = \frac{1}{n-1} \sum_{j \neq i}^{n} T_i^j$$

At the end of a task, all reputation values are calculated and are used as the weighted value for each node's decision. This can be written as $D = \sum_{i=1}^{n} T_i \cdot P_i^d$, for decision D, n number of nodes, and P_i^d as the decision of single node i.

4.3 EXPERIMENT

4.3.1 ARGoS Simulator

For the purpose of testing and evaluating ISRaft consensus protocol, the ARGoS simulator was used [124]. ARGoS is a swarm robotic simulator that is able to simulate large-scale swarms for any purpose. In our research, each robot acts as an independent ISRaft node, operating under the same guidelines detailed in the previous sections. Each node was given access to a running ISRaft network and was able to read and receive commands by having a unique identifier in the network.

A single node was able to execute commands either willingly (such as the heartbeat messages) or through a client controller running simultaneously. This was used for testing malicious activities such as message droppings, or messages that were not part of the original protocol.

The experiment was conducted as a simulated computer cluster with similar hardware to small mobile phones - a single core with 1.5GHz and 2 GB of memory. A total of 20 nodes operate with the ARGoS Simulator where the output data was carefully monitored by a 3rd party operator.

4.3.2 Setup

A total of 20 nodes were used in the experiment similar to [125]. The goal of this experiment was to help a set of autonomous units to make decisions about the colors of tiles in a 20×20 grid of black, white and gray tiles. The colors of these 400 tiles were randomly selected. For simplicity purposes, the nodes were able to communicate with each other freely, without distance restriction.

At the beginning of the experiment, each node was set to hold the data type that is relevant for this specific experiment. A 2D array was initialized as `char arr[20][400]` where the first dimension was the number of nodes in the cluster and the second di-

mension was the number of tiles in the system.

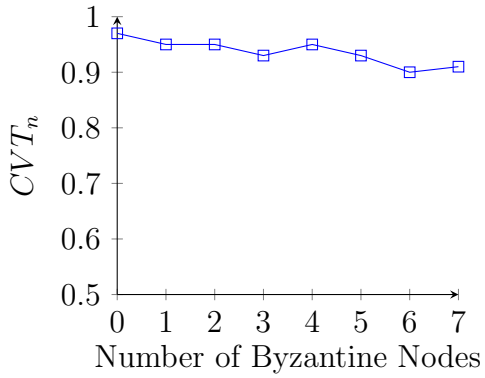
The experiment was set as follows: when a node stands on a colored tile, it reads that color. Upon success, the node then sends the signed *RequestAdd* RPC to the cluster including the tile number (1 to 400) and the color it reads. The leader receiving the RPC will then change the data of the 2D array at the location of the requesting node with the tile number and color reported. After some time, the leader will then send the *AppendBlock* RPC to the cluster with the new changes. A node receiving the RPC will then check to see if it has already read the color of that same tile. If yes, he will compare the results. If it is a correct color, the trust value of the reporting node will be increased, otherwise, it will be decreased. At the end of the experiment, the reputation value for each node is calculated and the decision for each tile is made based on the reputation values and reported colors. If, for example, tile 4 had 2 votes: a node with reputation 0.45 voted "Black" and a node with reputation 0.8 voted "White," the final decision will be "White."

During the experiment, Byzantine nodes were added to the cluster to see how the trust values are changed based on false messages. These Byzantine nodes always report wrong colors.

4.3.3 Technical Results

The experiment ran 35 times with different *election timeout* values ranging from 1 to 100 millisecond (assigned randomly) for a total of 5 minutes for each experiment. At the end of each run, we calculated the percentage of correctly voted tiles, noted as CVT_n for experiment n.

For each experiment with the same number of byzantine nodes, we averaged the CVT value to get these results:



As shown above, despite the decrease in CVT value when more byzantine nodes were added to the experiment, the overall value never went below 90%. This illustrates that the validation process along with the calculated trust values help in maintaining correctness of the model.

4.4 CONCLUSIONS AND FUTURE WORK

ISRaft is a consensus protocol that can be implemented among autonomous units in a secure data sharing environment. This protocol makes it possible to achieve consensus in the presence of adversarial nodes. Moreover, by using trust and reputation values, it becomes possible to validate the authenticity and correctness of the shared data. This can be easily implemented on resource-constrained devices and our model works perfectly in any infrastructure where regular encrypted communication methods can negatively affect the performance of the system.

In our future work, we will expand the implementation of ISRaft to deal with different tasks. We will also implement the protocol on real autonomous units to see how it operates in real-life scenarios.

CHAPTER 5

LOCALIZED STATE-CHANGE CONSENSUS (LSC)

5.1 INTRODUCTION

Blockchain technology began its popularity in 2008 with Satoshi Nakamoto's new peer-to-peer algorithm and his innovative way of achieving consensus among permissionless users called Proof-of-Work[22]. Ever since, more and more algorithms emerged with similar yet different methods of achieving consensus. Famous ones are the Proof-of-Stake[24], Proof-of-Authority and many more. Although Satoshi's initial motivation was to use the proposed algorithm as a digital currency, it has recently been implemented in many other fields, such as health care, supply chain, information sharing and more...

The Blockchain technology allows users to validate and secure immutable data which is replicated across the majority of users, giving the unique decentralization quality to it. Nowadays, most exchanges, whether it's a transaction or any other form of data, are being monitored and accepted by a trusted third party member (healthcare, banking, etc...). The reason is that if the exchange is not monitored, some people might share false data, making the exchange untrustable. The decentralized solution aims to solve this issue by executing a consensus algorithm designed to validate any report of data without the need for any third-party organization. More specifically, the data is stored on immutable blocks containing data. Every given time, a new block is added to the chain by an approval process called 'mining'. Different algorithms have a different mining process, however, the end goal is the same for all - the miner of the block gets rewarded. In some of these consensus

algorithms, the process and validation of data is pre-defined in a way that users can execute it automatically by running a simple server that can operate independently. This is why new blockchain applications have extended to autonomous fields such as robotic swarms and autonomous driving[120] where a communication is made between machines.

5.1.1 Decentralized Autonomous Units

The first time the blockchain data structure was introduced was in 1990 by Stuart Habert and W.Scott Strornetta [19] and was originally intended to timestamp digital documents making them tamper-proof. Over the years, blockchain data structure has expanded to many other fields, such as economy, e-voting, assembly line, etc. Many of these applications do not rely on cryptocurrency exchange, rather the users exchange information in a decentralized fashion. Different decentralized applications are classified as either public, federated/consortium or private. Public systems do not have any restriction on peers, and it doesn't require any authentication process for when joining the network or initiating trades. The public decentralized system is maintained only by the public community, which means a higher level of decentralization. Private, or permissioned system, operates under the leadership of a group often called 'consortium' and is the only group that can manage the system. Private decentralized system is a special type of permissioned system, where the network is managed by only one governing organization. These types of systems implement a registration process where a user has to go in order to be able to execute transactions.

Autonomous Units (AU) rely heavily on communication in order to operate properly, complete tasks and share information [119]. This is why a decentralized solution can be great for handling secured and immutable communication for AU. Autonomous units are required to be flexible and scalable, in order for them to operate in many fields and scenarios. Decentralized autonomous units can contain many individual

nodes and the loss of a few nodes should not affect the system, contributing to its robustness. This realization led to the emergence of more and more **Decentralized Autonomous Organizations** (DAO), or Decentralized Autonomous Corporation (DAC), which are blockchain-powered organizations that can operate independently without any central authority. Users can make decisions and are able to communicate in a decentralized fashion by a set of self-execution rules without the need for a mediator intervention.

Autonomous unit solutions are considered different from the crypto solutions that are mostly deployed on public decentralized systems. There are similar concepts such as the requirement for consensus and blockchain architecture, however there are many differences such as the requirement for the blockchain to be lightweight, having a secured way of communication, and having a control over the users who join and leave the network (permissioned). Some examples of unique AU tasks are exploration, surveillance, and data classification.

5.2 DECENTRALIZED SOLUTIONS FOR STATE CHANGE AND AUTONOMOUS SYSTEMS

In this section, we briefly review research works that proposed blockchain based mechanisms in autonomous and state change applications. ISRaft was first introduced in 2021[7] as a modified classical Raft [113] algorithm designed to handle Byzantine nodes and to improve the server-to-client communication from the original algorithm. Users would hold elections every fixed amount of time called *election-timeout*, and the selected leader would then have the responsibility of 'mining' the next block in the chain. LSC relies heavily on that idea, however, it differs greatly in its design and its state-change methodology. Other similar papers can be broken down into systems which rely on state change to operate successfully and systems which are usually run with autonomous units deployment.

Wang et al. [126] present an overview of the definition of Decentralized Autonomous Organization, and are setting standardization for DAO systems. This can be summed as '*Distributed and Decentralized*', '*Autonomous and Automated*' and '*Organized and Ordered*'. LSC successfully accomplished each of these characterizations by nature and therefore can be seen as a DAO subsystem we call Decentralized Autonomous Units, or DAU. The design of this consensus algorithm is meant to be thus implemented in autonomous units that require no human intervention such as communication between drones, smart cars or nanobots, which is required for the system to achieve an automated consensus in an organized fashion.

Moniz [127] presented a synchronized state machine consensus algorithm called Istanbul Byzantine Fault Tolerance (IBFT). It is a consensus algorithm designed to handle Byzantine-fault tolerant users in the Quorum blockchain. This algorithm is mostly used for state machine replication (SMR), namely achieving a consensus on a single state for multiple machines. Because of its low communication complexity, it can be considered a scalable solution. However, it is designed to only handle a single set of states, making it inflexible for different environments. Wei Ren [128] proposed a mechanism for achieving consensus among multi-vehicle systems. This consensus is designed to handle situations where only a portion of vehicles in the system can have access to a reference state. The consensus is achieved under the assumption of partitioning and can handle vehicles that cannot have access to the reference state.

Ferrer [29] describes how blockchain technology can be used as an innovative solution for deploying robotic swarms. This solution relies on the immutability property of the blockchain to solve problems that can occur in a non-blockchain deployment. This solution uses proof of work to achieve consensus, however, it can easily modify to fit other algorithms. Strobel et al. [125] designed a robotic swarm system that uses a blockchain approach in a collective decision-making scenario where a consensus is achieved on the total tile colors in an environment of black-and-white tiles. This

research is designed to handle byzantine-nodes by using blockchain-based smart contracts. The Byzantine nodes can easily be identified and excluded from the swarm. This research focuses primarily on achieving a consensus on a specific task and is deployed on the Ethereum network. Queralta et al. [32] improves the communication between autonomous units using blockchain technology. Proof-of-Work consensus is used to measure resources and the Proof-of-Stake is used to validate transactions. This design is suitable for either permissioned or permissionless blockchains with the use of a Single Longevous Blockchain that utilizes ad hoc collaboration in order to allow new nodes to enter the system. This design also includes a ranking mechanism similar to other trust models. However, nodes are not chosen based on their rankings or their level of trust.

5.3 NOTATIONS, DEFINITIONS, AND PROPERTIES

LSC is a *scalable decision-making election-based* consensus algorithm. This means that unlike many other blockchain implementations, which have mainly focused on achieving a consensus for the transactions on the ledger, LSC is designed to achieve consensus on the validity of the data that is written on the chain. In other words, LSC is designed to provide decision based consensus by validating the information sent between users.

This dissertation nonetheless describes LSC under the context of a *permissioned* blockchain. Users must have prior-knowledge of all other users and their signatures. When a new user joins the network it is then undergoing a series of enrollment processes, where it is assigned a pair of cryptographic keys and a trust value. Among the protocol's immutable communication, LSC also includes a state-change validation process where a group of users can change a state of some external information. It is then undergoing a validation process that includes the comparison of trust values and state reading. If the validation data is accepted by the majority, the state will change

and it will be reflected on the blockchain. Two major properties that are required for achieving the decision-based consensus are: **Authentication** and **Trust**.
Table 5.1 covers the notations we will be using for the rest of this section.

1. **Authentication:** Users are assigned a pair of cryptographic keys used for authentication of messages. When a user joins the network, it shares its public-key with all other users. When the user sends any message, it then signs it using its private-key in an asymmetric encryption fashion.
   Formally, for key-generator `G` and security parameter `k`, a public key (`pk`) and a corresponding private key (`sk`) are generated by:

   $$(pk, sk) \leftarrow G(1^k)$$

   Let S be a *signing* function that takes a private key (`sk`) and a string (`x`) and returns a tag value (`q`).
   Let `V` be a *verifying* function that takes a public key (`pk`), a string (`x`) and a tag value (`q`) and returns *accepted* if the tag value matches the expected value or *rejected* if not. Users verify messages (`x`, `q`) and reject any that does not include a valid signature. For the purpose of this model, any key-pair encryption algorithm, such as RSA, can be used.

2. **Trust:** Users hold a trust value `T` which is a numeric value that quantifies how trustworthy other users are. This value is defined by:

   Definition 5 (Reputation Value) *Let* $\mathtt{T}_\mathtt{i}^\mathtt{j}$ *be the trust value assigned by* $\mathtt{u}_\mathtt{j}$ *to* $\mathtt{u}_\mathtt{i}$. *Let* $\mathtt{T}_\mathtt{i}$ *be the reputation function that illustrate how trustworthy* $\mathtt{u}_\mathtt{i}$ *is:*

   $$T_i = \frac{1}{n-1} \sum_{j \neq i}^{n} T_i^j$$

   Users update the trust value of others when an interaction is made. These values are unique and do not have to be symmetric, ie., $\mathtt{T}_\mathtt{i}^\mathtt{j} \neq \mathtt{T}_\mathtt{j}^\mathtt{i}$. Trust value is used for data validation and state-change acceptance.

These two properties are what makes LSC functional in the presence of adversarial users. They help the network in achieving consensus for any state-change task it is given.

As stated earlier, LSC protocol is designed to handle state change consensus for an environment with multiple parameters, each with its own state and data. Users on the network are tasked with achieving consensus over different states. Assuming states can be changed by the environment, it is important for the LSC protocol to recognize any state-change and update it on the blockchain. A good example of such an environment will be a road system, where each road can be in either of three states: open, semi-open and closed. The state of any road can change depending on the amount of traffic it holds. Users can be cars driving and reading the environmental data of the roads. When a car recognizes a different state on one road, it can begin the state-change consensus to update the new state on the blockchain. Users who contribute and verify the state-change will be rewarded with increasing trust value.

5.4 OUR PROPOSED LOCALIZED STATE-CHANGE CONSENSUS MECHANISM

In this section, we cover the LSC consensus protocol as well as the 5 main communication algorithms. The main goal of LSC is to allow immutable and real-time decision-making between users on the LSC network. The second goal is to validate messages with adjustable trust values that determine the validity of the user.

LSC achieves consensus over a set of messages constructing agreed upon blocks through a Byzantine fault tolerant (BFT) protocol. This protocol supports message authentication, partitioning and fast computation, all of which makes it ideal for resource-constrained devices.

Table 5.1: Notation Used

Notation	Description
u_i	i^{th} user
T_i^j	Trust value assigned by j to i
ET_i	i^{th}Election Timeout
PK_{u_i}	Public key of user i
Sig_i	Signature of i
d_j^i	State of data point j as seen by user i
$M[Sig_i, d_j^i, t_1]$	Message M signed by i with state s
L	Number of nearest users
h	Number of signatures threshold

5.4.1 Design Overview

LSC consensus protocol is a method where all agents hold an agreed upon ledger containing data. This ledger is constructed of blocks forming a chain. An agent can be in either of the three main roles: (1) Follower, (2) Candidate, or (3) Leader. It is also possible to be an active validator, however, it is a temporary sub-role. A leader is the only authority that can concatenate new blocks to the chain, acting as the *miner* of the newly concatenated block. Followers and candidates can add new information to a new block by sending the data to the leader.

LSC aims at achieving consensus among agents in a highly dynamic environment with a set of n agents where $U = \{u_1, u_2, ..., u_n\}$. Communications among agents are

accomplished using five main algorithms. For the sake of simplicity, we assume that the communication is conducted on a secured channel without the possibility of having a man-in-the-middle attack. This, however, can be easily addressed in future works by using verifiable protocols. The five algorithms are:

- *RequestVote:* Initiated by a candidate agent and it is sent to all other agents. This message is sent as part of the *Election Process*.

- *StateChange:* Initiated by any agent and it is sent to `L` nearest agents that can validate a given message. This message consists of the data point ID, new state, timestamp, and signatures of agents approving the data. When a total of $\mathtt{h_i}$ agents sign the message sent by agent `i`, it is sent to the leader to be added to the next block.

- *AppendBlock:* Initiated by the leader at the end of every *leadership term*. This message contains the new block to be added to the chain. The agent receiving this message is required to respond with a signed approval message.

- *CommitBlock:* Initiated by the leader after receiving *AppendBlock* approval from the majority of agents.

- *Heartbeat:* Initiated by the leader and it is sent to all other agents. This message includes signed votes from the election, latest block number, and leadership term timer counting down to the end of the term. This message is sent periodically.

The number of required validators $\mathtt{h_i}$ for a *StateChange* message is defined as:

Definition 6 *Suppose* `n` *is the total number of agents in the network and* $\mathtt{T_i} \in [\mathtt{0}, \mathtt{1})$ *denote the reputation value of agent* `i`*. The number of required validators for the local StateChange message sent by agent* `i` *is calculated by:*

$$h_i = \lfloor \frac{n \times (1 - T_i)}{4} \rfloor \tag{5.1}$$

A message that is proved to be correct rewards the agent by increasing the agent's reputation value. These five algorithms make our proposed protocol easy to implement in almost any resource-restraint device. Figure 5.1 presents a flowchart of the leadership term process.

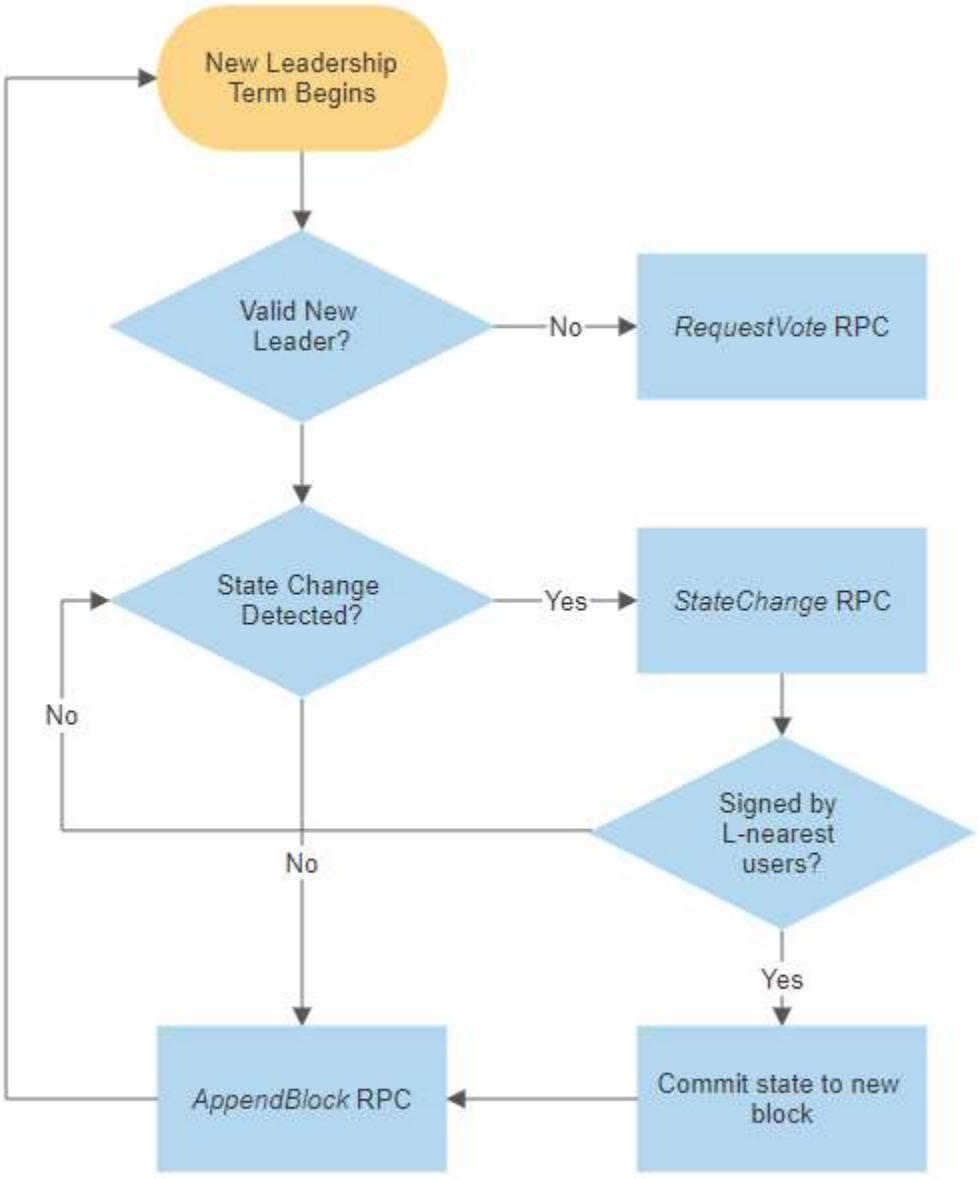


Figure 5.1: Leadership term flowchart.

5.4.2 Participants

There are four roles in the LSC network: follower, candidate, validator, and leader.

1. **Follower:** A follower is the basic and most common role in the LSC network. The followers' role is given automatically for any agent who joins the network after the registration process. Followers are what drives the system to a consensus by taking part in the election process as voters as well as by verifying any state-change event.

2. **Candidate:** Candidates' role is given to followers who initiated the *election process* after no *heartbeat* message has arrived over a fixed period of time, called *election timeout*, or when the *Leadership Term* has ended.

   - *Election Timeout:* Randomized local variable for any agent who joins the network.
   - *Leadership Term:* Fixed global variable

   Candidates compete to become the next leader, who is then rewarded with a significant increase in reputation value. Agents assign the candidate roles to themselves willingly when the conditions are met.

3. **Leader:** A leader is the highest authority who is responsible for sealing the next block on the LSC network. A leader is elected during the *Election* process, and there can only be one leader for every newly generated block. Since the leader cannot reasonably be expected to maintain a fully synchronized communication with all other agents, it is expected that the followers will be able to rebroadcast leader messages, excluding the heartbeat.

4. **Validator:** A validator's main purpose is to verify that a given data point's state has changed and that the *StateChange* message is true and valid. The process is fairly simple. Validators add their signature to the original *StateChange*

message and then share it with nearby agents who then also get the validator role.

This is a dynamic role and any agent can become a validator as long as it is close enough to the data point and to another validator agent.

5.4.3 Election Process

The first step of the protocol is to elect a leader in a *leader election* procedure. A leader is required to send periodic *Heartbeat* messages to all other agents in the network. This message contains the signed signatures from the latest election, latest block number, and leadership term timer. This message acts as a leadership proof. Agents receiving this message validate the votes and check if the latest block number is at least equal to the block number on their database. When the timer reaches 0 or when no heartbeat messages have arrived over the expected period, a new election process begins.

At the beginning of an election, a follower agent changes its state to a candidate, and he begins sending out signed *RequestVote* message, including the latest block number to all the agents it can contact. The latest block represents the term number of the chosen leader. For every new term, a new leader is elected. Agent $\mathtt{u}_{\mathtt{m}}$ who receives a *RequestVote* message will initially check the following conditions:

- The agent did not receive any *heartbeat* messages from the current leader.
- The candidate agent is not the leader of the current term.
- The block number from the *RequestVote* message is at least equal to the agent's latest block plus 1.
- The message has a valid signature.

If all the conditions are met, the voting agent will hold its vote for a fixed period of time equal to *Election Timeout* $\mathtt{e}$. During this time, if any other *RequestVote*

messages arrive, it will once again check if the conditions are met. If so, it will then compare the reputation value of the candidate agents. At the end of $\mathtt{e}$, the vote will be sent only to the agent with the higher reputation value. Algorithm 5 illustrates the voting response process for an agent after receiving a *RequestVote* message.

The process ends when a candidate receives the majority of the votes, that is, at least $n/2+1$ votes for a network of n agents. At that point, the elected leader starts sending out *heartbeat* messages to the network, thus completing the election process.

5.4.4 Data Point's State Architecture - Local Consensus

LSC runs on an environment with a set of $\mathtt{m}$ data points $\mathtt{S} = \{\mathtt{d_1}, \mathtt{d_2}, ..., \mathtt{d_m}\}$. These data points are given per environment and can be changed when redeploying the system in different environments. A single data point can be in any number of states. For the sake of simplicity, we will define these states as numerical values; however, it can be any data type. As in real life environments, data point's state can change independently and randomly by conditions that the system is not necessarily aware of. It is up to the system to recognize the change and validate the data on the blockchain. The process goes as follows:

1. An agent $\mathtt{u_k}$ recognizes a change in the state of some data points $\mathtt{d_j}$ in the environment.

2. $\mathtt{u_k}$ then sends a *StateChange* message to the $\mathtt{L}$ nearest agents, called **validators**. This message contains the data point ID, new state, and a timestamp.

3. Any other agent receiving the *StateChange* message can decide whether to add its signature or not. When an agent adds his signature, it then sends the newly signed *StateChange* message back to the sender and to his nearest $\mathtt{L}$ agents.

4. Steps 2 and 3 are repeated until a total of $\mathtt{h_i}$ agents sign the original *StateChange* message. The value of h is calculated by the reputation values of the agents

Algorithm 5 RequestVote Response

Require: *RequestVote* Message (RV_i) and Agent (u_i)

Ensure: Accept or Reject RV

$t \leftarrow 0$
$RV \leftarrow RV_i$
$Decrease_reputation(T_i^m)$
while $t \neq e$ **do** ▷ Election Timeout
    if $RV.block_num <= len(chain)$ **then**
        Reject RV
    else if $V(u_i.pk, RV, S(RV)) == reject$ **then**
        Reject RV
    end if
    if new RV_j from agent u_j **then**
        $Decrease_reputation(T_j^m)$
        if $T_i^m > T_j^m$ **then** ▷ Compare reputation
            Reject RV_j
        else
            $RV \leftarrow RV_j$
        end if
    end if
    $t \leftarrow t + 1$
end while
Accept RV

and by the total number of agents (Definition 6).

5. The $\mathtt{h_i}$-th agent sends the message to the elected leader to add it to the next block.

6. Once a new block is mined, all other agents in the system update the state of data point $\mathtt{d_j}$ to the new state.

This process happens every time an agent recognizes a data point's state that is different from the state written on the blockchain. Agents who validate the state change and add a signature to the original *StateChange* message are called *validators*, and are rewarded with reputation value upon a successful state change. The number of validators for each state-change is given at the design level, and can be changed based on the reputation value of the sender, whenever the system redeploys, or by forking. The reason for this localized consensus is to prevent overloading the leaders with any state change and to prevent a possible Distributed Denial of Service (DDoS) attack. Algorithm 6 covers the process of *StateChange* message signature. This process can also be referred to as the localized consensus process.

Agents near any state change will also likely to sign and send many *StateChange* messages from different agents who repeatedly send the signed message. If the message has already been signed by the recipient, he will neither sign, nor send it again. Another possible scenario is to have multiple $\mathtt{h_i}$ length signed messages of the same origin that are sent to the leader. In this case, the leader will only add the first message and will reject any message with the same origin. This process can be represented by an $\mathtt{h_i}$ tree structure in which each node is an agent who signed the message and sent it to $\mathtt{L}$ other nodes. Figure 5.2 illustrates an example related to this matter.

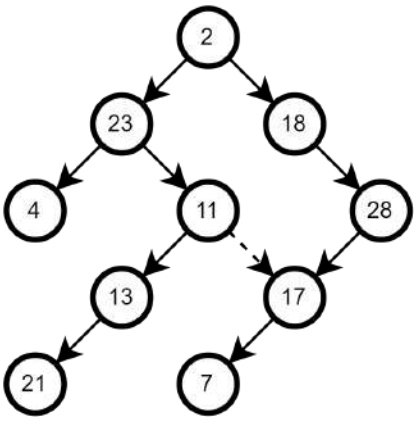


Figure 5.2: Representation of L tree structure

Here, $\mathtt{L} = 2$ meaning that at most 2 neighboring agents can validate a message, and a total of $h_{14} = \lfloor \frac{30 \times (1-0.333)}{4} \rfloor = 5$ signatures are required to validate any messages sent by agent 14, where $n = 30$ and $T_{14} = 0.333$. The numbers within each node represent an agent ID. We can see that there are 3 valid tree paths: $14 \rightarrow 23 \rightarrow 11 \rightarrow 13 \rightarrow 21$, $14 \rightarrow 18 \rightarrow 28 \rightarrow 17 \rightarrow 7$ and $14 \rightarrow 23 \rightarrow 11 \rightarrow 17 \rightarrow 7$. In this example, agent number 17 signed 2 different chains. This is valid since it is the first signature on either path. The first path to arrive at the leader is likely to be the one added on the next block.

5.4.5 Block Generation/Mining

Once a leader has been elected for some term t, it begins the preparations of adding a new block to the chain. At the end of every *leadership term*, a new block is required to be appended to the chain in a process called Block Generation or Mining. Every block on the chain has the following data:

- **Block Number:** Also represents the term number of the current block.
- **Block Leader:** The agent who mined the block.
- **Timestamp:** The time when the block was mined.

Algorithm 6 StateChange Signature

Require: *StateChange* Message M

Ensure: Signed message or Reject

1: **if** $ifV(pl, M, S) = rejected$ **then** ▷ Sig Validation
2: Exit
3: **end if**
4: $t \leftarrow 0$
5: **while** $t! = mt$ **do** ▷ Message timeout
6: **if** $d_i == M[d_i]$ **then return** $M[\text{Sig}, d_i, t]$
7: **end if**
8: read d_i ▷ Keep reading while message not timedout
9: $t++$
10: **end while**

- **State Changes:** Any data point that has been changed is added to the block. A list of state changes with the original message and signatures is added.
- **Previous Block Hash:** The hash value of the previous block.

The blocks are hashed and connected by the appropriate block number and the hashing value of the previous block. There can only be one leader for any single block. Followers can add data to the block by sending a signed *StateChange* message consisting of a list of the new data point's states. Each of these items holds the data point ID, new state value of the data point, original author of the message, and validator's signatures. The number of signatures per message is defined by $\texttt{h}_\texttt{i}$ and can be different among agents based on their reputation values. Agents with higher reputation values will have to provide more signatures, hence, more validators are required to verify the authenticity of the message. A leader will add the state-change message to the block if and only if:

- Enough validators have included their signatures.
- The originator of the message hasn't already been included in the block.

There can be a case where the leader does not get any *statechange* messages throughout his *leadership term*. In this case, the leader will simply mine an *empty* block that does not contain any state change data. Figure 5.3 shows an example for block properties in the LSC.

Once a block is ready to be appended at the end of the *leadership term*, the leader sends the *AppendBlock* to all agents in the network. This message includes the next new block as well as the signed votes from the election in which it won. This prevents anyone from trying to disguise themselves as the leader. When an agent receives the *AppendBlock* message, it will check the following:

- Votes are legitimate.
- Hash value of the previous block and the block number fits the latest block in the chain. If not, the agent will update the chain from nearby agents.

If the conditions are met, the agent will send a signed *AppendBlock* approval back to the leader. If and when the majority of agents in the network send the approval, the leader can then send the *CommitBlock* message including the signed approvals. Only upon receiving these messages, agents can commit the new block to the chain.

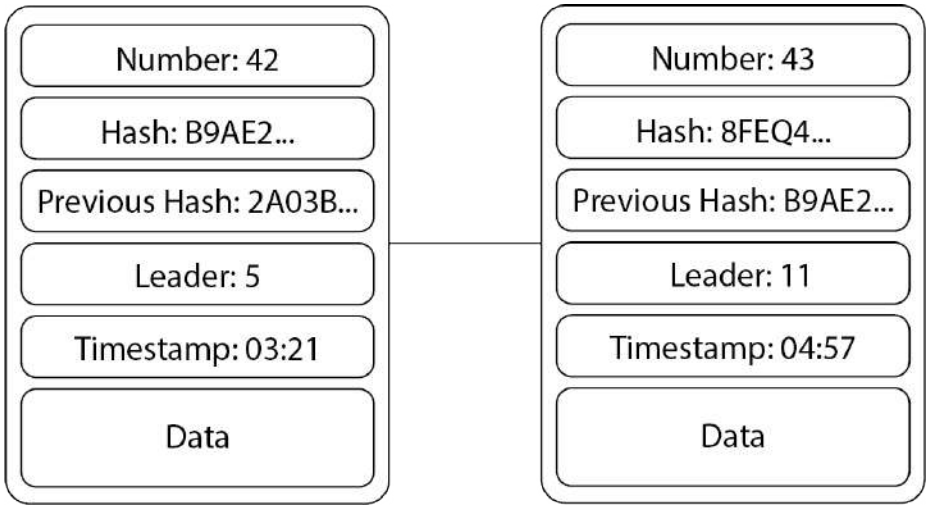


Figure 5.3: Blocks example in the LSC infrastructure.

5.4.6 Trust and Reputation

Reputation plays a significant role in verifying messages and it can be calculated for a single agent by averaging the trust values of all other agents. Reputation value can be any decimal number in the interval $[0, 1]$ (in our implementation, we set it to have a total of 4 decimals) and it represents the confidence in which agents rely on each other, i.e., 0 being non-trusted and 1 being highly-trusted. These values are unique and do not have to be symmetrical, i.e., for agents $\mathtt{u_i}$ and $\mathtt{u_j}$, $\mathtt{T_i^j} \neq \mathtt{T_j^i}$ [129].

When a follower sends a message to any other agent in the LSC network, the recipient immediately decreases the reputation value of the sender by a small amount, called **reputation cost**. However, once the message has been verified and added to the blockchain, the sender's reputation value increases by a margin larger than the cost. This is to prevent agents from sending unauthorized messages and to prevent possible DDoS attacks on the network. Agents constantly change the reputation values of other agents over the network based on interactions in the network. Table 5.2 covers the reputation change for different messages in detail.

Table 5.2: LSC messages and their reputation costs.

Message	reputation Cost	Details
RequestVote	✗	Candidates send this message during the election process. This does not cost any reputation since we want all agents to have a chance of winning without the risk of constantly losing on reputation values.
StateChange	✓	Initiated and sent to `L` nearest agents, called **validators**. When a validator does not approve the state change, either by observing or by insufficient reputation values, it will decrease the reputation value of the origin agent as well as agents who signed it. However, if the recipient approves the message and **signs** it, it will increase the reputation value.
AppendBlock	✓	This message is initiated by the leader and is part of the mining block. This message includes the signed votes from the latest election. Agents will decrease the reputation value of the sender in the case if the signed votes are **not** valid.
CommitBlock	✓	This message can only be initiated by the leader after the *AppendBlock* message was successful. Agents will decrease the reputation value of the sender in the case the included approvals are **not** valid.
Hearbeat	✗	This periodic message does not affect the reputation value.

Once a new block is committed on the blockchain, the reputation values of all agents who originated and signed a *StateChange* message on the new block increases as a reward in a process, called **Block Reputation Increment** where all agents update the state of the changed data point on the new block.

5.5 TECHNICAL ANALYSIS OR OUR PROPOSED SOLUTION

We now evaluate the effectiveness and scalability of LSC followed by security analyses.

5.5.1 Effectiveness

The effectiveness of the protocol can be measured by the total amount of messages that are sent among different agents on the network. LSC proposes a simple five-message protocol to limit the total required bandwidth and memory per agent. The most commonly used message over a group of agents is the *StateChange* message that requires the signature of $\texttt{h}_\texttt{i}$ validators. With the total number of validators and the total number of nearest agents $\texttt{L}$, we can easily calculate the maximum number of messages required for a single validation process for agent i.

$$\sum_{k=1}^{h_i} L^k = \frac{L - L^{h_i+1}}{1 - L} \tag{5.2}$$

For example, with $\texttt{L} = 2$ and $\texttt{h}_\texttt{i} = 5$, the maximum number of messages to be sent during this validation process is 62. To calculate the time complexity of the total required *StateChange* messages, we will start by setting $\texttt{h}_\texttt{i}$ to be the expansion value from Def. 6. Since we know that $\texttt{L} > 1$, we can calculate the time complexity to be $O(L^{h_i})$, which is the expected complexity from the *geometric series* and it depends on the height of the tree, noted as $\texttt{h}_\texttt{i}$.

5.5.2 Scalability - A Storage Analysis

The generic assumption of the blockchain is that each agent can read a committed block and update the data point with the new state data. When a new block is

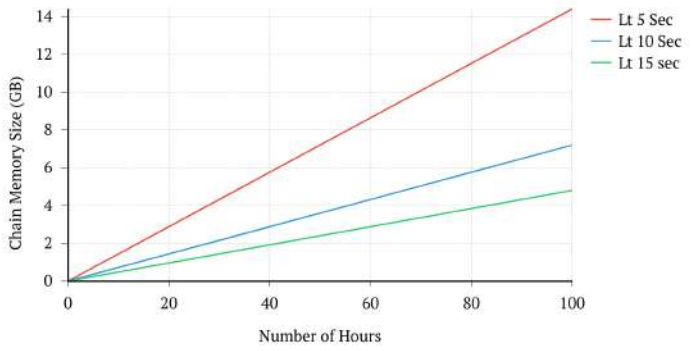


Figure 5.4: Required memory for chains with different `Lt` values.

committed, it is under the assumption of the LSC protocol that each agent stores the new block along with all previous blocks back to genesis. This assumption, however, can have some issues especially when working with systems that can have limited memory capacity. This issue creates a scalability issue if the system is required to operate for a long period of time with many agents.

The memory requirement of each agent depends on the state-change content, the number of state changes per block, and how frequently a new block is mined. More agents means more validators for any state change. As definition 6 shows, an increase of `n` increases the value of $\mathtt{h_i}$.

Let us denote the size of the state change message as $\mathtt{St_w}$, number of state changes per block as `St`, and the weight of the block header as $\mathtt{H_w}$. A single block weight $\mathtt{B_w}$ can be calculated as follows [130]:

$$B_w = \sum_{j=0}^{\mathrm{St}} \mathrm{St}_w^j + H_w \tag{5.3}$$

Next, from experimental evaluation, we know that the weight of a single *StateChange* message ($\mathtt{St_w}$) is $\approx$ 1 KB, header ($\mathtt{H_w}$) is $\approx$ 2 KB, and an average of 200 *StateChange* messages can be added per block. This value is averaged by the amount of data a single agent can receive and handle. From (5.1), we can then calculate the total memory requirement for a single block to be $\approx$ 200 KB. Next, let us denote the

length of the leadership term as `Lt`. This value depicts how fast a new block is added to the chain. Different `Lt` values can drastically increase or decrease the weight of the LSC chain. Figure 5.4 shows the exponential growth in memory size when increasing `Lt`. Agents are then required to hold more than 2 GB of chain data per day on average, which can cause a problem for resource-constrained devices that are required to operate for a long period of time. This problem, however, can be addressed in either two ways:

1. Store the blockchain data on a dedicated cloud server. Agents can operate by utilizing their private-keys for reading and mining. The cloud data can be hashed and stored for chain validation processes.

2. Set a group of dedicated full-node agents who hold the full chain, unlike regular agents who hold only a snapshot of the chain.

5.5.3 Security Analysis

This section described security analysis of LSC while also explaining the different security restrictions that have been added to the protocol. For this protocol to function, we have to assume there are enough honest users on the network. This number depends on the total number of users, as well as the average number of validators per message. Below is the formal definition of the honest majority assumption.

Definition 7 (Honest Majority) *Suppose* `n` *total number of users on the network,* `T` *is the average reputation value of the network and* `H` *is the average number of validators per StateChange message. The total number of malicious users on the network* Γ *is calculated by:*

$$\Gamma < (n - T) \times (1 - \frac{H}{n})$$

False Validation Attack. Validators are essential for the protocol to function properly. The number of validators per *StateChange* message h is determined by the

reputation value and the total number of users. This value is usually small, since it only requires a local consensus. This means that a small number of users can gather and send a false state change message only to have their trust value increase in the Block Reputation Increment process. However, it can be easily detected as the data is written on the chain, and can be accessed by anyone when a malicious activity is suspected. Once recognized, it is then easy to isolate and disregard any previous data made by that group. An immediate solution to this attack can be assigning a special trusted validator to a given set of data points. This validator's signature is required for any *StateChange* message that is sent regarding any data point in the designated set. It can easily be implemented in a trusted blockchain environment.

Partitioning. Partitioning can happen when a group of users is separated from the main group, by natural or malicious means. The partitioned group can miss a new leader election, new blocks committed and state changes. This problem, however, is solvable by design. A partitioned group of users will not receive any *heartbeat* messages, causing them to start a separated leader election process, followed by a separated state change and block mining. Once the partition is removed, a user in the partitioned group might see two different chains - the one from the main group and the other from the partitioned group. A user will **always** follow the chain with the higher block number, so any state changes that were added to the shorter blockchain will be removed. Any data point's state that has been removed is likely to be added at a later point, as long as the state is different from the latest state on the chain.

Block Withholding Attack. Block withholding can occur when the current leader does not publish the latest block in time. Once a new block is mined, participants can update the data point's state to reflect the state on the blockchain. Validators are also rewarded with increased trust value. When a leader holds the block, he is taking a risk of having his trust value reduced significantly, making it unlikely for him to ever be elected as the leader again. Block withholding within the

network can be easily detected by any user who does not receive an *AppendBlock* message within the leadership term period. The effect of not publishing a block in time can be a momentary delay in the state update, however, the state will be updated on the next block with a different leader.

Impersonation Attack. Impersonating a user on the network can have a major negative effect. It can change the leader votes, unvalidate data and more. However, this attack is not possible in the LSC because of the nature of the permissioned network and the registration process. Any communication is required to be signed with the sender's private key `pk`. This prevents any malicious impersonation activity. On the other hand, if a user gets a hold of a different user's private key, it can easily impersonate that other user. This problem is true for any other system that relies on private-public key encryption.

Distributed Denial of Service Attack. Distributed Denial of Service attack is a very common attack on systems that rely on communication. In general, this attack is an adversarial attempt to disrupt the normal activity of other users by sending frequent messages to overwhelm and cause a traffic jam in the communication. In LSC, any message is being 'priced' with some trust value. This means that a malicious user's trust value will decrease as long as it continues sending messages. Once a trust value reaches the lower threshold for a different user, any other message sent by him will be immediately dropped, stopping it from overwhelming the network.

CHAPTER 6
CONCLUSIONS

This section summarizes the contribution accomplished in this dissertation. Two innovative consensus algorithms were presented that are used for achieving consensus on information shared between party members. We have studied the problem of implementing decentralized solutions in a dynamic environment with dynamic state-change data points, and we presented the idea of 'localized consensus' to achieve a faster partial consensus over a subset of users in the network.

1. Paper [7] presents a modification of Raft consensus algorithm for information sharing between users while handling malicious nodes. Digital signature is used with encrypted PRCs to support operative communication. ISRaft is implemented in C programming language and tested on multiple servers.

2. Paper [8] expands the original ISRaft to support implementation in autonomous units in a secure data sharing environment. This protocol utilizes trust and reputation values and can validate the shared data by comparing it between users on the network.

3. Paper [9] presents a new information consensus algorithm for an immense and highly dynamic environment using localized consensus. Similar to the previous ISRaft, this consensus algorithm is also capable of validating and authenticating the content of the shared data by first initiating a local consensus using cryptographic communication means including trust and reputation values. The new consensus algorithm, LSC, assures confidentiality, integrity and validity of messages on the blockchain among all agents.

6.1 FUTURE WORK

This dissertation presents two innovative consensus algorithms called ISRaft and Localized State-Change (LSC). Both have many applications and can be implemented in many fields. Combining these algorithms with Autonomous units has proven to be extremely compatible due to the nature of autonomous units and their applications. While much of the work for this dissertation is already completed, there is still significant work to be made. This section presents some topics that could be of interest for future research.

- The proposed schemes use a basic definition of trust and reputation values, however this can be expanded to include other protocols for stronger trust and reputation computation. Also, we did not discuss possible attacks on these trust and reputation systems. It is also possible to implement solutions that make it have a stronger resistance to these types of attacks.

- There is a potential for using secure multi-party computation to build fully localized consensus protocols. LSC uses an L-tree data structure to support localized consensus, yet it is possible to share the information via MPC thus adding another layer of security to the shared data.

- The implementations presented in this dissertation are on a small scale and operate in a 'clean' environment. It is important to test the capabilities of the algorithms on a large scale implementation that includes a dynamic environment and malicious nodes.

- Many blockchain implementations use cryptographic primitives that are not quantum resistance (QR). ISRaft and LSC consensus use digital signatures, for example, that are considered less secure in the presence of quantum computers. It is possible to improve the quantum security of the proposed schemes by

integrating primitives that are quantum safe, such as the Quantum Digital Signature (QDS).

BIBLIOGRAPHY

[1] Nojoumian M, Golchubian A, Njilla L, Kwiat K, Kamhoua C. Incentivizing blockchain miners to avoid dishonest mining strategies by a reputation-based paradigm. In: Computing Conference. Springer; 2018. p. 1118-34.

[2] Lamport L, Fischer M. Byzantine generals and transaction commit protocols. Technical Report 62, SRI International; 1982.

[3] Yu F, Ruan N, Cheng S. Rational Manager in Bitcoin Mining Pool: Dynamic Strategies to Gain Extra Rewards. In: Proceedings of the 15th ACM Asia Conference on Computer and Communications Security; 2020. p. 572-83.

[4] Dorri A, Steger M, Kanhere SS, Jurdak R. Blockchain: A distributed solution to automotive security and privacy. IEEE Communications Magazine. 2017;55(12):119-25.

[5] Si H, Sun C, Li Y, Qiao H, Shi L. IoT information sharing security mechanism based on blockchain technology. Future Generation Computer Systems. 2019;101:1028-40.

[6] Ølnes S, Ubacht J, Janssen M. Blockchain in government: Benefits and implications of distributed ledger technology for information sharing. Elsevier; 2017.

[7] Zamir L, Nojoumian M. Information sharing in the presence of adversarial nodes using raft. In: Proceedings of the Future Technologies Conference. Springer; 2021. p. 159-72.

[8] Zamir L, Shaan A, Nojoumian M. ISRaft consensus algorithm for autonomous units. In: 2021 IEEE 29th International Conference on Network Protocols (ICNP). IEEE; 2021. p. 1-6.

[9] Zamir L, Nojoumian M. Localized State-Change Consensus in Immense and Highly Dynamic Environments. Cryptography. 2022;6(2):23.

[10] Li S, Lin B. Accessing information sharing and information quality in supply chain management. Decision support systems. 2006;42(3):1641-56.

[11] Deters FG, Mehl MR. Does posting Facebook status updates increase or decrease loneliness, An online social networking experiment. Social psychological and personality science. 2013;4(5):579-86.

[12] Dinev T, Hart P, Mullen MR. Internet privacy concerns and beliefs about government surveillance–An empirical investigation. The Journal of Strategic Information Systems. 2008;17(3):214-33.

[13] De Kruijff J, Weigand H. Understanding the blockchain using enterprise ontology. In: International Conference on Advanced Information Systems Engineering. Springer; 2017. p. 29-43.

[14] Weber I, Xu X, Riveret R, Governatori G, Ponomarev A, Mendling J. Untrusted business process monitoring and execution using blockchain. In: International Conference on Business Process Management. Springer; 2016. p. 329-47.

[15] López-Pintado O, García-Bañuelos L, Dumas M, Weber I. Caterpillar: A Blockchain-Based Business Process Management System. In: BPM (Demos); 2017. .

[16] Mendling J, Weber I, Aalst WVD, Brocke JV, Cabanillas C, Daniel F, et al. Blockchains for business process management-challenges and opportunities. ACM Transactions on Management Information Systems (TMIS). 2018;9(1):1-16.

[17] Nojoumian M. Rational trust modeling. In: International Conference on Decision and Game Theory for Security. Springer; 2018. p. 418-31.

[18] Harding J, Powell G, Yoon R, Fikentscher J, Doyle C, Sade D, et al. Vehicle-to-vehicle communications: readiness of V2V technology for application. United States. National Highway Traffic Safety Administration; 2014.

[19] Haber S, Stornetta WS. How to time-stamp a digital document. In: Conference on the Theory and Application of Cryptography. Springer; 1990. p. 437-55.

[20] Zheng Z, Xie S, Dai H, Chen X, Wang H. An overview of blockchain technology: Architecture, consensus, and future trends. In: 2017 IEEE international congress on big data (BigData congress). Ieee; 2017. p. 557-64.

[21] Dwork C, Naor M. Pricing via processing or combatting junk mail. In: Annual international cryptology conference. Springer; 1992. p. 139-47.

[22] Nakamoto S, et al.. Bitcoin: A peer-to-peer electronic cash system.(2008); 2008.

[23] Fullmer D, Morse AS. Analysis of difficulty control in bitcoin and proof-of-work blockchains. In: 2018 IEEE Conference on Decision and Control (CDC). IEEE; 2018. p. 5988-92.

[24] King S, Nadal S. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. self-published paper, August. 2012;19:1.

[25] Duong T, Chepurnoy A, Fan L, Zhou HS. Twinscoin: A cryptocurrency via proof-of-work and proof-of-stake. In: Proceedings of the 2nd ACM Workshop on Blockchains, Cryptocurrencies, and Contracts; 2018. p. 1-13.

[26] Hardjono T, Lipton A, Pentland A. Toward an interoperability architecture for blockchain autonomous systems. IEEE Transactions on Engineering Management. 2019;67(4):1298-309.

[27] Baza M, Nabil M, Lasla N, Fidan K, Mahmoud M, Abdallah M. Blockchain-based firmware update scheme tailored for autonomous vehicles. In: IEEE Wireless Communications and Networking Conference; 2019. p. 1-7.

[28] Wang Y, Su Z, Zhang K, Benslimane A. Challenges and solutions in autonomous driving: A blockchain approach. IEEE Network. 2020;34(4):218-26.

[29] Ferrer EC. The blockchain: a new framework for robotic swarm systems. In: Future Technologies Conference. Springer; 2018. p. 1037-58.

[30] Strobel V, Dorigo M. Blockchain technology for robot swarms: A shared knowledge and reputation management system for collective estimation. In: 11th International Conference on Swarm Intelligence. vol. 11172. Springer; 2018. p. 425-6.

[31] Singh PK, Singh R, Nandi SK, Ghafoor KZ, Rawat DB, Nandi S. An efficient blockchain-based approach for cooperative decision making in swarm robotics. Internet Technology Letters. 2020;3(1):e140.

[32] Queralta JP, Westerlund T. Blockchain-powered collaboration in heterogeneous swarms of robots. arXiv preprint arXiv:191201711. 2019.

[33] Badea L, Mungiu-Pupazan MC. The economic and environmental impact of bitcoin. IEEE Access. 2021;9:48091-104.

[34] Rizun PR. A transaction fee market exists without a block size limit. Block Size Limit Debate Working Paper. 2015:2327-4697.

[35] Saleh F. Blockchain without waste: Proof-of-stake. The Review of financial studies. 2021;34(3):1156-90.

[36] Nair PR, Dorai DR. Evaluation of performance and security of proof of work and proof of stake using blockchain. In: 2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV). IEEE; 2021. p. 279-83.

[37] Larimer D. Delegated proof-of-stake (dpos). Bitshare whitepaper. 2014;81:85.

[38] Castro M, Liskov B, et al. Practical byzantine fault tolerance. In: OsDI. vol. 99; 1999. p. 173-86.

[39] Lewenberg Y, Bachrach Y, Sompolinsky Y, Zohar A, Rosenschein JS. Bitcoin mining pools: A cooperative game theoretic analysis. In: Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems; 2015. p. 919-27.

[40] Jesus EF, Chicarino VR, De Albuquerque CV, Rocha AAdA. A survey of how to use blockchain to secure internet of things and the stalker attack. Security and Communication Networks. 2018;2018.

[41] Kiffer L, Levin D, Mislove A. Stick a fork in it: Analyzing the Ethereum network partition. In: Proceedings of the 16th ACM Workshop on Hot Topics in Networks; 2017. p. 94-100.

[42] Buterin V, et al. Ethereum white paper: a next generation smart contract & decentralized application platform. First version. 2014;53.

[43] Vasek M, Thornton M, Moore T. Empirical analysis of denial-of-service attacks in the Bitcoin ecosystem. In: International conference on financial cryptography and data security. Springer; 2014. p. 57-71.

[44] Swanson E. Bitcoin mining calculator; 2013.

[45] Johnson B, Laszka A, Grossklags J, Vasek M, Moore T. Game-theoretic analysis of DDoS attacks against Bitcoin mining pools. In: International Conference on Financial Cryptography and Data Security. Springer; 2014. p. 72-86.

[46] Laszka A, Johnson B, Grossklags J. When bitcoin mining pools run dry. In: International Conference on Financial Cryptography and Data Security. Springer; 2015. p. 63-77.

[47] Luu L, Saha R, Parameshwaran I, Saxena P, Hobor A. On power splitting games in distributed computation: The case of bitcoin pooled mining. In: 2015 IEEE 28th Computer Security Foundations Symposium. IEEE; 2015. p. 397-411.

[48] Rosenfeld M. Analysis of bitcoin pooled mining reward systems. arXiv preprint arXiv:11124980. 2011.

[49] Bag S, Sakurai K. Yet another note on block withholding attack on bitcoin mining pools. In: International Conference on Information Security. Springer; 2016. p. 167-80.

[50] Bag S, Ruj S, Sakurai K. Bitcoin block withholding attack: Analysis and mitigation. IEEE Transactions on Information Forensics and Security. 2016;12(8):1967-78.

[51] Solat S, Potop-Butucaru M. Zeroblock: Timestamp-free prevention of block-withholding attack in bitcoin. arXiv preprint arXiv:160502435. 2016.

[52] Eyal I. The miner's dilemma. In: 2015 IEEE Symposium on Security and Privacy. IEEE; 2015. p. 89-103.

[53] Kwon Y, Kim D, Son Y, Vasserman E, Kim Y. Be selfish and avoid dilemmas: Fork after withholding (faw) attacks on bitcoin. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security; 2017. p. 195-209.

[54] Lee S, Kim S. Countering block withholding attack efficiently. In: IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). IEEE; 2019. p. 330-5.

[55] Eyal I, Sirer EG. Majority is not Enough: Bitcoin Mining is Vulnerable. arXiv e-prints. 2013.

[56] Heilman E. One weird trick to stop selfish miners: Fresh bitcoins, a solution for the honest miner. In: International Conference on Financial Cryptography and Data Security. Springer; 2014. p. 161-2.

[57] Zhang R, Preneel B. Publish or perish: A backward-compatible defense against selfish mining in bitcoin. In: Cryptographers' Track at the RSA Conference. Springer; 2017. p. 277-92.

[58] Kogias EK, Jovanovic P, Gailly N, Khoffi I, Gasser L, Ford B. Enhancing bitcoin security and performance with strong consistency via collective signing. In: 25th {usenix} security symposium ({usenix} security 16); 2016. p. 279-96.

[59] Bai Q, Zhou X, Wang X, Xu Y, Wang X, Kong Q. A deep dive into blockchain selfish mining. In: ICC 2019-2019 IEEE International Conference on Communications (ICC). IEEE; 2019. p. 1-6.

[60] Eyal I, Sirer EG. Majority is not enough: Bitcoin mining is vulnerable. In: International conference on financial cryptography and data security. Springer; 2014. p. 436-54.

[61] Sapirshtein A, Sompolinsky Y, Zohar A. Optimal selfish mining strategies in bitcoin. In: International Conference on Financial Cryptography and Data Security. Springer; 2016. p. 515-32.

[62] Negy KA, Rizun PR, Sirer EG. Selfish mining re-examined. In: International Conference on Financial Cryptography and Data Security. Springer; 2020. p. 61-78.

[63] Grunspan C, Pérez-Marco R. Profit lag and alternate network mining. arXiv preprint arXiv:201002671. 2020.

[64] Bastiaan M. Preventing the 51%-attack: a stochastic analysis of two phase proof of work in bitcoin. In: Availab le at http://referaat. cs. utwente. nl/conference/22/paper/7473/preventingthe-51-attack-a-stochasticanalysis-oftwo-phase-proof-of-work-in-bitcoin. pdf; 2015. .

[65] Dey S. Securing majority-attack in blockchain using machine learning and algorithmic game theory: A proof of work. In: 2018 10th computer science and electronic engineering (CEEC). IEEE; 2018. p. 7-10.

[66] Bae J, Lim H. Random mining group selection to prevent 51% attacks on bitcoin. In: 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W). IEEE; 2018. p. 81-2.

[67] Bala R, Manoharan R. Security enhancement in Bitcoin protocol. In: 2018 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET). IEEE; 2018. p. 1-4.

[68] Rosenfeld M. Analysis of hashrate-based double spending. arXiv preprint arXiv:14022009. 2014.

[69] Pérez-Solà C, Delgado-Segura S, Navarro-Arribas G, Herrera-Joancomartí J. Double-spending prevention for bitcoin zero-confirmation transactions. International Journal of Information Security. 2019;18(4):451-63.

[70] Karame GO, Androulaki E, Capkun S. Double-spending fast payments in bitcoin. In: Proceedings of the 2012 ACM conference on Computer and communications security; 2012. p. 906-17.

[71] Hern A. Bitcoin currency could have been destroyed by 51% attack. The Guardian. 2014;16.

[72] Kroll JA, Davey IC, Felten EW. The economics of Bitcoin mining, or Bitcoin in the presence of adversaries. In: Proceedings of WEIS. vol. 2013; 2013. p. 11.

[73] Thetangle.org - iota tangle explorer and statistics. In: Thetangle.org;. .

[74] Bahack L. Theoretical bitcoin attacks with less than half of the computational power (draft). arXiv preprint arXiv:13127013. 2013.

[75] Memon M, Bajwa UA, Ikhlas A, Memon Y, Memon S, Malani M. Blockchain beyond Bitcoin: block maturity level consensus protocol. In: 2018 IEEE 5th International Conference on Engineering Technologies and Applied Sciences (ICETAS). IEEE; 2018. p. 1-5.

[76] Heilman E, Kendler A, Zohar A, Goldberg S. Eclipse attacks on bitcoin's peer-to-peer network. In: 24th {USENIX} Security Symposium ({USENIX} Security 15); 2015. p. 129-44.

[77] Apostolaki M, Zohar A, Vanbever L. Hijacking bitcoin: Routing attacks on cryptocurrencies. In: 2017 IEEE Symposium on Security and Privacy (SP). IEEE; 2017. p. 375-92.

[78] Bonneau J, Miller A, Clark J, Narayanan A, Kroll JA, Felten EW. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In: 2015 IEEE symposium on security and privacy. IEEE; 2015. p. 104-21.

[79] Okupski K. Bitcoin developer reference. In: Eindhoven; 2014. .

[80] Marcus Y, Heilman E, Goldberg S. Low-Resource Eclipse Attacks on Ethereum's Peer-to-Peer Network. IACR Cryptol ePrint Arch. 2018;2018:236.

[81] Nayak K, Kumar S, Miller A, Shi E. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE; 2016. p. 305-20.

[82] Gervais A, Ritzdorf H, Karame GO, Capkun S. Tampering with the delivery of blocks and transactions in bitcoin. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security; 2015. p. 692-705.

[83] Khalilov MCK, Levi A. A survey on anonymity and privacy in bitcoin-like digital cash systems. IEEE Communications Surveys & Tutorials. 2018;20(3):2543-85.

[84] Grunspan C, Pérez-Marco R. On profitability of stubborn mining. arXiv preprint arXiv:180801041. 2018.

[85] Castro M, Druschel P, Ganesh A, Rowstron A, Wallach DS. Secure routing for structured peer-to-peer overlay networks. ACM SIGOPS Operating Systems Review. 2002;36(SI):299-314.

[86] Singh A, Castro M, Druschel P, Rowstron A. Defending against eclipse attacks on overlay networks. In: Proceedings of the 11th workshop on ACM SIGOPS European workshop; 2004. p. 21-es.

[87] Singh A, et al. Eclipse attacks on overlay networks: Threats and defenses. In: In IEEE INFOCOM. Citeseer; 2006. .

[88] Sit E, Morris R. Security considerations for peer-to-peer distributed hash tables. In: International Workshop on Peer-to-Peer Systems. Springer; 2002. p. 261-9.

[89] Bakker A, Van Steen M. Puppetcast: A secure peer sampling protocol. In: 2008 European Conference on Computer Network Defense. IEEE; 2008. p. 3-10.

[90] Bortnikov E, Gurevich M, Keidar I, Kliot G, Shraer A. Brahms: Byzantine resilient random membership sampling. Computer Networks. 2009;53(13):2340-59.

[91] Jesi GP, Montresor A, van Steen M. Secure peer sampling. Computer Networks. 2010;54(12):2086-98.

[92] Saad M, Cook V, Nguyen L, Thai MT, Mohaisen A. Partitioning attacks on bitcoin: Colliding space, time, and logic. In: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS). IEEE; 2019. p. 1175-87.

[93] Back A, et al. Hashcash-a denial of service counter-measure. 2002.

[94] Bedi HS, Roy S, Shiva S. Game theory-based defense mechanisms against DDoS attacks on TCP/TCP-friendly flows. In: 2011 IEEE symposium on computational intelligence in cyber security (CICS). IEEE; 2011. p. 129-36.

[95] Feinstein L, Schnackenberg D, Balupari R, Kindred D. Statistical approaches to DDoS attack detection and response. In: Proceedings DARPA information survivability conference and exposition. vol. 1. IEEE; 2003. p. 303-14.

[96] Hyvärinen H, Risius M, Friis G. A blockchain-based approach towards overcoming financial fraud in public sector services. Business & Information Systems Engineering. 2017;59(6):441-56.

[97] Wang A, Mohaisen A, Chen S. An adversary-centric behavior modeling of DDoS attacks. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). IEEE; 2017. p. 1126-36.

[98] Lau F, Rubin SH, Smith MH, Trajkovic L. Distributed denial of service attacks. In: Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0. vol. 3. IEEE; 2000. p. 2275-80.

[99] Feder A, Gandal N, Hamrick J, Moore T. The impact of DDoS and other security shocks on Bitcoin currency exchanges: Evidence from Mt. Gox. Journal of Cybersecurity. 2017;3(2):137-44.

[100] Eyal I, Sirer EG. How to disincentivize large bitcoin mining pools. Blog post: http://hackingdistributed com/2014/06/18/how-to-disincentivize-large-bitcoin-mining-pools. 2014.

[101] Saad M, Thai MT, Mohaisen A. POSTER: deterring ddos attacks on blockchain-based cryptocurrencies through mempool optimization. In: Proceedings of the 2018 on Asia Conference on Computer and Communications Security; 2018. p. 809-11.

[102] Belotti M, Kirati S, Secci S. Bitcoin pool-hopping detection. In: 2018 IEEE 4th International Forum on Research and Technology for Society and Industry (RTSI). IEEE; 2018. p. 1-6.

[103] Lee S, Kim S. Pooled Mining Makes Selfish Mining Tricky. IACR Cryptol ePrint Arch. 2018;2018:1230.

[104] Luu L, Velner Y, Teutsch J, Saxena P. Smartpool: Practical decentralized pooled mining. In: 26th {USENIX} Security Symposium ({USENIX} Security 17); 2017. p. 1409-26.

[105] Singh SK, Salim MM, Cho M, Cha J, Pan Y, Park JH. Smart contract-based pool hopping attack prevention for blockchain networks. Symmetry. 2019;11(7):941.

[106] Salimitari M, Chatterjee M, Yuksel M, Pasiliao E. Profit maximization for bitcoin pool mining: A prospect theoretic approach. In: 2017 IEEE 3rd international conference on collaboration and internet computing (CIC). IEEE; 2017. p. 267-74.

[107] Tang C, Wu L, Wen G, Zheng Z. Incentivizing honest mining in blockchain networks: a reputation approach. IEEE Transactions on Circuits and Systems II: Express Briefs. 2019;67(1):117-21.

[108] Macrinici D, Cartofeanu C, Gao S. Smart contract applications within blockchain technology: A systematic mapping study. Telematics and Informatics. 2018;35(8):2337-54.

[109] Christidis K, Devetsikiotis M. Blockchains and smart contracts for the internet of things. Ieee Access. 2016;4:2292-303.

[110] Mauri L, Cimato S, Damiani E. A Comparative Analysis of Current Cryptocurrencies. In: ICISSP; 2018. p. 127-38.

[111] Baruffaldi G, Sternberg H. Chains in chains-logic and challenges of blockchains in supply chains. 2018.

[112] Mettler M. Blockchain technology in healthcare: The revolution starts here. In: 2016 IEEE 18th international conference on e-health networking, applications and services (Healthcom). IEEE; 2016. p. 1-3.

[113] Ongaro D, Ousterhout J. In search of an understandable consensus algorithm. In: 2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14); 2014. p. 305-19.

[114] Nakamoto S. Bitcoin: A peer-to-peer electronic cash system. Manubot; 2019.

[115] De Angelis S, Aniello L, Baldoni R, Lombardi F, Margheri A, Sassone V. Pbft vs proof-of-authority: applying the cap theorem to permissioned blockchain. eprints, University of Southampton. 2018.

[116] Pease M, Shostak R, Lamport L. Reaching agreement in the presence of faults. Journal of the ACM (JACM). 1980;27(2):228-34.

[117] Copeland C, Zhong H. Tangaroa: a byzantine fault tolerant raft. Tech. Rep; 2016.

[118] Bamakan SMH, Motavali A, Bondarti AB. A survey of blockchain consensus algorithms performance evaluation criteria. Expert Systems with Applications. 2020:113385.

[119] Brambilla M, Ferrante E, Birattari M, Dorigo M. Swarm robotics: a review from the swarm engineering perspective. Swarm Intelligence. 2013;7(1):1-41.

[120] Yuan Y, Wang FY. Towards blockchain-based intelligent transportation systems. In: IEEE 19th International Conference on Intelligent Transportation Systems; 2016. p. 2663-8.

[121] Zamir L, Nojoumian M. Information Sharing in the Presence of Adversarial Nodes Using Raft. In: Future Technologies Conference; 2021. .

[122] Nojoumian M, Lethbridge TC. A New Approach for the Trust Calculation in Social Networks. In: E-business and Telecommunication Networks: 3rd International Conference on E-Business, Best Papers. vol. 9 of CCIS. Springer; 2008. p. 64-77.

[123] Nojoumian M, Stinson DR. Social Secret Sharing in Coud Computing Using a New Trust Function. In: 10th IEEE Annual International Conference on Privacy, Security and Trust; 2012. p. 161-7.

[124] Pinciroli C, Trianni V, O'Grady R, Pini G, Brutschy A, Brambilla M, et al. ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. Swarm intelligence. 2012;6(4):271-95.

[125] Strobel V, Castelló Ferrer E, Dorigo M. Managing Byzantine Robots via Blockchain Technology in a Swarm Robotics Collective Decision Making Scenario. In: International Conference on Autonomous Agents and MultiAgent Systems. ACM; 2018. p. 541–549.

[126] Wang S, Ding W, Li J, Yuan Y, Ouyang L, Wang FY. Decentralized autonomous organizations: concept, model, and applications. IEEE Transactions on Computational Social Systems. 2019;6(5):870-8.

[127] Moniz H. The Istanbul BFT consensus algorithm. arXiv preprint arXiv:200203613. 2020.

[128] Ren W. Multi-vehicle consensus with a time-varying reference state. Systems & Control Letters. 2007;56(7-8):474-83.

[129] Nojoumian M, Lethbridge TC. A new approach for the trust calculation in social networks. In: International Conference on E-Business and Telecommunication Networks. Springer; 2006. p. 64-77.

[130] Biswas S, Sharif K, Li F, Maharjan S, Mohanty SP, Wang Y. PoBT: A lightweight consensus algorithm for scalable IoT business blockchain. IEEE Internet of Things Journal. 2019;7(3):2343-55.