2024

Multi-Enclave Blockchain Architecture Integrating Useful Work For Member Selection In Committee-Based Consensus

Shawn C. Adams
University of Alabama at Birmingham

MULTI-ENCLAVE BLOCKCHAIN ARCHITECTURE INTEGRATING USEFUL WORK FOR MEMBER SELECTION IN COMMITTEE-BASED CONSENSUS

by

SHAWN C. ADAMS

YULIANG ZHENG, COMMITTEE CHAIR
PURUSHOTHAM BANGALORE
RAGIB HASAN
RAMARAJU RADARARAJU
CHENGCUI ZHANG

A DISSERTATION

Submitted to the graduate faculty of The University of Alabama at Birmingham,
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

BIRMINGHAM, ALABAMA

2024

MULTI-ENCLAVE BLOCKCHAIN ARCHITECTURE INTEGRATING USEFUL WORK FOR MEMBER SELECTION IN COMMITTEE-BASED CONSENSUS

SHAWN C. ADAMS

COMPUTER SCIENCE

ABSTRACT

Blockchains offer an alternative approach to centralized systems for data and record storage by using independently operated nodes in a distributed and decentralized network. Within a blockchain network, each node maintains an identical version of all data, without being directly controlled or managed by any centralized entity. Nodes maintain the synchronization of data through the network's consensus mechanism, a protocol built into the nodes to determine which node is given the right to append data to the network. Once chosen, the data added by the node is propagated to other nodes which update their local records to maintain synchronization. Common consensus mechanisms used fall into the categories of Proof of Work (PoW), Proof of Stake (PoS), and committee-based Byzantine Agreement (BA), each with their own respective strengths and weaknesses. Problems with consensus mechanisms can lead to scalability issues which can impact widespread adoption. This thesis proposes the Federated Advanced Work Adversarial Consensus (FAWAC) architecture as a novel blockchain network design using a multi-enclave, committee-based consensus mechanism that is energy efficient, resistant to centralization and achieves fast transaction processing. FAWAC incorporates components of PoW, PoS and BA in an interconnected multi-chain structure which capitalizes on the security benefits of PoW, and the speed benefits of PoS and BA. Energy inefficiency and centralization concerns are minimized through a novel approach for committee selection which uses the completion of useful work jobs as a substitute for traditional computational puzzles used by PoW without slowing down transaction processing. The enclaves are interconnected via a shared back end utilizing the Interplanetary File System (IPFS) connections on participating nodes which actively monitor both enclaves.

Keywords: Blockchain, Interplanetary File System, Consensus Mechanisms, Network Architecture, Distributed Computation

DEDICATICATION

Dedicated to my loving wife and children for their patience and understanding across the years of late nights and missed events while I worked towards this achievement.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ASIC	Application Specific Integrated Circuit
BFT	Byzantine Fault Tolerance
CID	Content Identifier
DAPP	Distributed Application
DPoS	Delegated Proof of Stake
DHT	Distributed Hash Table
ECDSA	Elliptic Curve Digital Signature Algorithm
EVM	Ethereum Virtual Machine
FAWAC	Federated Advanced Work Adversarial Consensus
IoT	Internet of Things
IPFS	Interplanetary File System
ML	Machine Learning
PoA	Proof of Authority
PoS	Proof of Stake
PoUW	Proof of Useful Work
PoW	Proof of Work

LIST OF DEFINITIONS

Application Specific Integrated Circuit (ASIC)	A specialized set of computer chips designed to have algorithm instructions directly imprinted on the chip to allow for faster computation of problems using the algorithm.
Adversarial Consensus	An approach to committee-based decision voting where the leader may purposefully propose invalid data to test if other participants are actively validating the data before voting.
Block	A data structure created by a node in a blockchain network which contains digitally signed transactions which have been validated and chosen to be included as part of a blockchain.
Blockchain	A network where transaction data is processed and recorded into blocks of data that are time-stamped and use hashing technology to create a sequential link between blocks created and stored by nodes individually maintaining identical copies of the ledger.
Cryptocurrency	A type of digital currency that is created as either a fuel for the network or prize/incentive awarded to those who validate data and add to the ledger.
Delegated Proof of Stake (DPoS):	A consensus algorithm where nodes use their respective stake as a proportional vote for a user who will perform validation on the end user's behalf.
Distributed Hash Table (DHT)	A system which uses hashing of values to create key-value pairs to identify the location of the data in a distributed system.
Distributed Ledger	A data structure that records transactions in such a way that it is immutable, not stored within a central location or controlled by a centralized entity and capable of maintaining multiple branches of records at the same time.
Hard Fork	In a blockchain network, when the version software/protocol is updated in such a way that nodes using the old version and the new version are not compatible and

	as such cannot communicate. The result is two distinct networks that share a common history up to the point of fork but are distinct after that point.
Hash Rate	The total hashing power of all nodes within a PoW system working to solve a cryptographic puzzle. Typically, the value is measured in terms of hashes per second.
Internet of Things (IoT)	The interconnection via the internet of computing devices embedded in everyday objects, enabling them to send and receive data.
Interplanetary File System (IPFS)	A distributed storage network that uses content-based addressing and distributed hash tables to provide access to the data in a peer-to-peer way.
Lazy Agreement	A behavior in a committee-based consensus system where a node votes to accept the proposed data without performing any validation.
Merkle Tree	A Hash structure wherein two inputs are used to produce a hash of both values. At the leaf level, each pair is hashed together, resulting in a new hash. The process is repeated until only a single hash exists.
Node	A computing device that exists as part of a distributed network by executing the required source code and network protocols.
Oracles	Programs which interact with a blockchain network by injecting data from outside sources enabling nodes on the network to consume the data.
Permissioned Blockchain	A category of blockchain technology that uses access control methods to restrict which nodes write information stored in the ledger.
Permissionless Blockchain	A category of blockchain technology where no access control methods are used to restrict which nodes write information to the ledger. As a result, anyone can join the network and participate in the consensus mechanism.
Private Blockchain	A category of blockchain technology that uses access control methods to restrict which users can read information stored in the ledger.
Proof of Authority (PoA)	A consensus mechanism which is achieved primarily through proving that a node has the appropriate permissions and authority to perform validation efforts. This approach typically is used in private or permission-based network where all nodes can be authenticated by a centralized system prior to performing validation activities.

Proof of Stake (PoS)	A consensus algorithm wherein the probability of being selected as the node to append data is proportional to the amount of the network's native currency the node controls.
Proof of Work (PoW)	A consensus algorithm wherein a difficult to solve, but easily verifiable problem is used to determine which node can append the next set of data.
Proof of Useful Work (PoUW):	A consensus algorithm wherein the difficult to solve problem involves providing a solution to a computationally intense program which has real-world implications. Examples could include updating a machine learning model or calculating values for protein folding.
Public Blockchain	A category of blockchain technology where no access control methods are used to restrict the ability of users to read information stored in the ledger.
Soft Fork	In a blockchain network, when the version of software/protocol is updated in such a way that nodes using the new version are backwards compatible to the old version. Nodes running the newer version will be able to validate blocks based on both the new and old validations rules but nodes running the old version will only be able to validate based on the old rules and ignore blocks following the new validation rules.
Smart Contract	A programmable piece of code that can exist on a distributed ledger system that is capable of automatically performing actions based on the state of input.
Temporary Fork	In a blockchain network, when two different blocks are validated at the same time and are considered valid by other nodes that the original validating nodes communicate with. Both are considered valid until the next block is validated and appended first, at which time, the one appended because the valid block and the other is discarded.
Token	A virtual representation of an asset that is digitally owned and managed through a smart contract.
Transaction	A digitally signed record added to a blockchain that changes ownership of a digitized asset or updates the state of a smart contract.
Wallet	An application which maintains a user's private key, allowing the user to create transactions and access blockchain data.

Web3	A term to describe applications which present some or all interactive front-end elements to a user through interaction with a blockchain back-end for data.
Validator	A participant in a consensus system where the member selected to record information to the ledger validates data being added and then inserts it into the ledger.

CHAPTER 1

INTRODUCTION

Human society can be viewed as an interconnected web of interactions between the members that make up the society. In general, people have come to rely on interactions with each other as part of their day to day lives. The items that an individual person owns are likely to have been purchased rather than created by that individual from scratch. Even if the individual created an something from scratch, the raw resources were likely acquired through some interaction and exchange with other members of society. The way in which people interact with each other to exchange goods and services has evolved over time from direct bartering between peers, to indirect exchanges facility by centralized entities. However, such centralization can place individuals at risk due to potential opportunities for censorship by the centralized entities. Additionally, the exchange of value between individuals on a world-wide scale relies on infrastructure that may result in delayed payments as the exchange is processed between multiple centralized agencies which may only have limited regional control to facilitate the exchange. These downsides have led to a resurgence of interest in systems that enable peer-to-peer exchanges that are resistant to centralization, and take advantage of updated infrastructures that enable near real-time settlement anywhere in the world.

Blockchain is a technology that has risen as the primary alternative to centralized systems. A blockchain is a network where transaction data is processed and recorded into blocks of data that are ordered using time-stamps and hashing technology to create a linked list that is stored identically on many distributed nodes across a network. Although not referred to as a blockchain initially, a good example of blockchain technology is the Bitcoin [59] network. Bitcoin was created as a system to enable the peer-to-peer transfers of value between two parties in a way that was decentralized, trust-less and secured using cryptographic technologies. The foundations outline in the Bitcoin white paper created the foundation for blockchain technology. Blockchain technology exists where technologies such as digital signatures, hashing, and network communication overlap with a system to achieve agreement between distributed nodes. A visual representation of the integrated technologies that make up blockchain can be seen in Figure 1.1. nicknamed as such due to the way the use of cryptographic hashes created an immutable chain of block data structures recording all accepted transaction within the network.

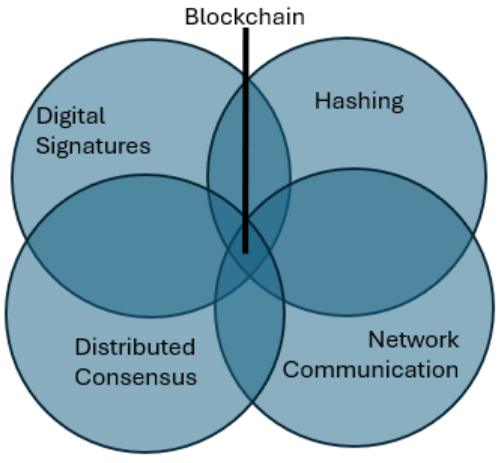

FIGURE 1.1: Overlap of technologies that make up blockchain

While the Bitcoin network only focused on the use case of peer-to-peer transfers of value, it was not long before others began to see the expanded potential of blockchain technology. In 2015, the Ethereum [16] network was presented as an extension of the capabilities of the Bitcoin network by adding programmable logic that could be stored and executed by nodes on the network. Ethereum introduced the concept of a "Smart Contract", immutable programs that can be interacted with through transactions processed by the network. The ability of Ethereum smart contacts to store and process data as well as hold value has opened blockchain technology into a wide range of use cases. Nearly any use case that can be created with a centralized system can also be created as a distributed smart contract. Some examples of smart contract uses cases include asset tokenization, and decentralized finance (DeFi) [6, 16], supply chain management [45, 57] smart cities [26, 62] and many more applications just to name a few. Even emerging ideas are being examined for potential blockchain integration. As more and more devices become connected to the internet, the concept of the Internet of Things (IoT) has recently become popular along with the idea of using blockcahin to provide a method of access control for IoT devices [52, 69].

Just as the concept of a blockchain was created by combining existing technologies on a novel way, blockchain continues to evolve through the incorporation of new technologies or new approaches. For example, the Interplanetary File System (IPFS) [14] network is collection of nodes developed to facility distributed content-based storage as an alternative to traditional location-based addressing. The technology operates as its own functional system without specific considerations for integration with blockchain, yet has used as a way to increase data storage for smart contracts by storing the data on IPFS and maintaining reference as pointers on chain [6, 7, 27, 67]. Additionally, IPFS has been fully integrated into

a previously created blockchain network which was designed to pay participants for participating in a distributed storage system called FileCoin [15]. While IPFS and blockchain seem like two technologies that would have uses cases that would sync up naturally, up until now, the integration has been limited. Among other things, one of the core goals of this work is to demonstrate system that more closely aligns the two technologies, capitalizing on the strengths of both.

1.1 RESEARCH HYPOTHESIS

While existing blockchain networks have made significant improvement in terms of speed and throughput, many still suffer from issues and concerns related to centralization. Not only do some consensus mechanism trend toward centralization over time as some nodes gain more power than other nodes, but the storage requirements of maintaining an ever growing list of blocks can act as a barrier for entry for some nodes without sufficient storage resources, leading to less nodes and more centralization. In addition, many blockchain networks are not taking advantage of new technologies that can help enable distribution of data in a way that improves connectivity between blockchain network with systems that exist outside the blockchain network. This work is centered on the idea that through a novel architecture, blockchain networks can become less resource intensive for nodes, promote fair and centralization-resistant selection of nodes to propose block while maintaining the security and integrity of the network. The hypothesis of this dissertation is as follows:

Our research effort involve the creation of a prototype blockchain network using the

Blockchain technology can be further decentralized and secured through the use of IPFS to create an interconnected multi-enclave system, enabling safe and verifiable committee selection for committee-based consensus to improve the security and efficiency of blockchain networks.

Federated Advanced Work Adversarial Consensus (FAWAC) architecture design, to validate the hypothesis to show the feasibility of such a blockchain network to compete with existing solutions. Through experimentation we are able to validate the viability of using a three enclave structure to introduce the security of work-based consensus with the speed and centralization resistance of committee-based consensus. Further, we show that the traditional wasteful nature of work-based consensus mechanisms can be improved through the application of generalized, yet useful problems that could benefit from being executed in a distributed fashion.

In addition, we show that the work being completed in our distributed system can be used as prerequisite for nodes to participate as committee members. By adding the requirement, the network becomes resistant to Sybil attacks, wherein an attacker attempts to gain power by creating multiple nodes on the network, by imposing a cost for the nodes to be considered for committee membership. While it may not be possible to eliminate the presence of all malicious nodes, we also demonstrate that their presence can be minimized through a reputation-based selection system that can identify dishonest behavior performed by nodes selected for validation committees. We show that the mechanism to identify dishonest behavior by committee members is able to reduce the probability of the node being selected in the future, providing an additional layer of security for the network. Finally, we expand upon our concept of generalized code execution to show that this approach can support an

alternative approach for smart contract development compatible with the unique network design.

1.2 RESEARCH QUESTIONS

This work works as an analysis of the following research questions that were considered during prototype development. Below each question is a high level summary of the findings from experiments during previous research efforts and how the results tie into the overall FAWAC architecture. Later chapters will discuss the concepts in greater detail.

RQ1: Can arbitrary code be constructed and validated in such a way that can be determined to be safe to execute on a remote node? Can we validate the code was actually executed by a remote node?

Programs which take a long time to execute or require resources which an end user may not want to dedicate on a local machine are the primary candidates for the type of code which would benefit from execution on remote nodes. However, any code that comes from sources that are not trusted may have the potential to be malicious and damage node performing the execution. In addition, nodes may submit results that are inconsistent with the actual expected results if the node is acting maliciously. Since a node is submitting the program for execution, the node is not likely to know the correct answer ahead of time, making it difficult to differentiate between honestly calculated and fabricated results. In this work, we present a solution to both problems that were validated based on previous prototypes and experiments. Previously, we developed a parser that was able to identify specific indicators of potentially dangerous code.

Further, our previous research demonstrated the ability to validate deterministic code through the use of IPFS for storage and a comparison of a hash of the result. With IPFS, data stored is hashed to create a Content Identifier (CID) which is used to access the file. If the content of a file changes, when it is added to IPFS, it will generate a new CID. Similarly, if the same file is added by different nodes, regardless of location, the same CID will be generated for the file.

RQ2: Can arbitrary code be efficiently packaged by one node and be properly reconstructed for execution by a set of remote nodes?

The previous question focused on determining if arbitrary code is safe to execute on the front end and if the results can be verified as correct on the back end. However, the question of whether the execution of the arbitrary code is possible still needs to be considered to fill in the gap between the front end and back end of the process. The creation of arbitrary programs may have dependencies that are not readily apparent in order to enable successful execution on another device which may be set up differently. For example, the syntax for Python 2 and Python 3 programming languages may allow some statements to execute in either language and produce the same result, Python 3 by itself is not backwards compatible and not all commands that do run on both produce the same results. Our architecture takes advantage of the results from our previous research to incorporate a mechanism to recreate and execute code on a remote system by using IPFS for storage and distribution of the files.

RQ3: Can IPFS be used efficiently as a shared-backend for data stored on the blockchain? To what extent can data be pruned from nodes and still have the entire history available? What limitations may be encountered when using IPFS?

Content based addressing is used for data stored on IPFS [14]. The data is broken into 256KB blocks, hashed, and stored across the network and can be found using a hash table lookup. Large files can be reconstructed and accessed using the unique CID generated for the file. When an IPFS node requests a file using the CID, it downloads the file locally and begins serving the file for other nodes also trying to access it. However, not every node automatically connects with every other node and the lookup process could be slow depending on the number of connections from the requesting node to one that hosts the content. During our previous research, we identified the limitation related to finding files if nodes are not connected and created a solution wherein nodes connect to other nodes in the network directly using peer connection requests. We incorporate a protocol used by the nodes that involves the transmission of the IPFS peer connection data for nodes on a committee, allowing nodes to directly set up a peer connection and reducing latency within the network. When a new committee is chosen, the connection information for all committee members are stored as data within the blocks, allowing for easy lookup and connection to the nodes. Nodes which are part of a committee have an incentive to maintain block data validated by the committee because these blocks contain references to their rewards for participation in the committee. Since data persists within IPFS as long as at least one node maintains a copy, nodes that are not part of the committee are able to prune data to reduce storage requirements without impacting the security of the network.

RQ4: Is it possible to actively identify malicious behavior on a committee? Can the identification of malicious behavior be used to limit potential selection for future committees? Can we actively identify and mitigate lazy agreement?

Faulty or malicious behavior by committee members in a blockchain using committee-based consensus can negatively impact the network and may lead to disruption of the blockchain. The recovery and resistance to such behavior is often referred to as a measure of the network's Byzantine Fault Tolerance (BFT). In typically committee-based approaches, as long as no more than one-third of nodes on the committee are faulty or malicious the consensus mechanism should still continue to operate as expected. However, if the threshold is reached consensus may not be reached. For this reason it is important to identify malicious committee members and decrease the probability of their selection in order to secure the network. We show that through our reputation-based system, malicious behavior can be identified and penalized, while honest behavior can be rewarded. Through a committee selection mechanism that takes into account random selection that has a bias towards higher reputation scores, malicious node selection is minimized over time. We have used experiments from previous research efforts to validate that the malicious node representation can be decreased over time.

Identification of malicious behavior by nodes creates a scenario where most of the nodes being selected are likely to be honest. This creates an interesting dynamic where an opportunity exists for nodes to assume the committee will be honest and vote to accept proposed blocks instead of actually performing validation. We have we dub this behavior "lazy agreement" and have identified in initial experiments that the behavior was not identified as malicious as part of the committee selection algorithm when the committee is made up of mostly honest nodes. However, the problem with lazy agreement is that it can lead to invalid data being accepted if the number of

malicious nodes and lazy nodes make up more than half the committee size. If a malicious block proposer is on a committee where more than half the committee is made up of lazy and malicious nodes, the invalid block can be accepted as valid. This can even occur if the malicious node is the only malicious node in the committee and the remaining members required to make up half of the committee are lazy. Essentially, lazy nodes amplify the block proposers power by voting to accept by default. We incorporate the "Adversarial Consensus" model as a way to combat lazy agreement. Adversarial Consensus allows a block producer to propose invalid data purposefully to test whether other nodes are actually performing validation. Through Adversarial Consensus, the presence of lazy nodes and malicious nodes are further reduced from committee selection over time.

RQ6: Can the FAWAC architecture support smart contract execution?

While many blockchain implementations currently exist which only support transactions which exchange value between end users, this covers only one of the many use cases which are possible when a blockchain incorporates smart contract capabilities. Most blockchain networks that support smart contracts are based on the model used by Ethereum, the FAWAC architecture has many changes that make it incompatible with other networks built on the Ethereum model. As such, a new solution for the storage of smart contract state data and execution of code needed to be developed from scratch. We demonstrate that the FAWAC architecture is able to support smart contract code stored within the blockchain in a way that appears to be comparable with alternative solutions.

1.3 ORGANIZATION OF THIS DISSERTATION

In this chapter (Chapter 1), we have presented the introduction to the research problem that our work address as well as provide the overall thesis statement. Chapter 2 presents a deeper dive into the background for blockchains, consensus mechanisms, smart contracts and IPFS. Additionally it examines the details for existing research efforts that are closely related to our own and their respective limitations. We provide a high level overview of the FAWAC architecture in Chapter 3. Chapter 4 examines the portion of the FAWAC framework that enables the safe and verifiable remote code execution required as part of the prerequisite criteria for the committee-based consensus mechanism. In Chapter 5 we begin to look at another core requirement of the FAWAC architecture, the ability to store blockchain data on IPFS so that data can be easily exchanged between the three enclaves. It describes the prototype developed to verify that a blockchain could be created that uses IPFS for data storage. Additionally the chapter describes how a multi-block structure is used to separate proposed block data, validation data and state updates, supporting the reputation-based committee selection mechanism. The chapter also outlines the structure for transactions, blocks and smart contracts. We explore the reputation based committee selection mechanism using the Adversarial Consensus model in Chapter 6. Adversarial Consensus is a mechanism that addresses risks introduced by nodes which vote to accept without performing validation first by penalizing the nodes for the activity. Finally we conclude this document with a review of the main concepts with a focus on the contributions made by this work and highlighting areas for future work.

CHAPTER 2

BACKGROUND

Blockchain is a technology that has only started to be used an alternative to existing currency systems in 2009, making it relatively new compared to existing currency technologies. It involves the integration of multiple technologies in order to provide a secure record of transfers between individuals. Just as blockchain can be seen as an evolutionary step for how humans use currency, advancements such as smart contracts can be seen as a further evolution of blockchain. The Federated Advanced Work Adversarial Consensus (FAWAC) architecture that this dissertation describes is a further evolutionary step for blockchain with a focus on improving fairness, maintaining security and providing additional benefits to users looking for distributed computation resources. As there are many interrelated technologies that are used to build blockchain and the FAWAC architecture, this chapter will provide background into those technologies to set the foundation for how FAWAC capitalizes on the benefits of each to create an improved architecture for blockchain networks.

2.1 BLOCKCHAIN PRINCIPALS

Blockchain is a technology that was originally created to support a decentralized digital currency that could act as an alternative to existing centralized currency systems. While specific features for a blockchain may be related to the specific implementation, in general, blockchain refers to a technology which enables the storage of permanent and immutable

data across a network of distributed and decentralized nodes. Nodes rely on the network's respective consensus mechanisms to maintain consistency and many existing networks have demonstrated the ability to act as alternative to centralized data-storage solution. The first network recognized as a blockchain was the Bitcoin [59] network and was first proposed in 2009.

Bitcoin was created to facilitate peer-to-peer transfers of value between individuals by keeping a decentralized digital ledger that is secure and immutable. The blockchain structure used by Bitcoin incorporates concepts from several fields of study into a single technology that enable it to act as distributed ledger system with the properties of being decentralized, immutable, tamper resistant and publicly verifiable [10, 69]. Bitcoin involved a careful integration of digital signatures using cryptographic keys, hashing to ensure data integrity, gossip protocols for propagation of messages across networks and even techniques for denial of service prevention [8].

Using public/private key pairs as a means for digital signatures to validate the authenticity of a sender was first explored in the late 1970s and has been improved upon with multiple updated versions since [4, 30]. Most blockchain implementations currently rely on an implementation of the Elliptic Curve Digital Signature Algorithm (ECDSA) [43] for digital signatures and the SHA family of hashing algorithms to create digital fingerprints of the data, though other methods can be used. Similarly network communication and sequentially time stamping data was a core pillar of research that the Bitcoin network built off of [59] and the original concept of Bitcoin's consensus mechanism, was based on a denial of service prevention system, called Hashcash [8].

A blockchain follows a cyclical process wherein individual nodes across the network collect and validate transactions before storing them in a block. As individual nodes may

produce different sets of transactions in different orders, the process of determining which node's block will be accepted as the universally accepted one by all nodes in the network, achieving consensus between the nodes. Specific approaches to achieve consensus are described in further detail in Section 2.2. A simplified illustration of the generalized block production cycle can be seen in Figure 2.1.

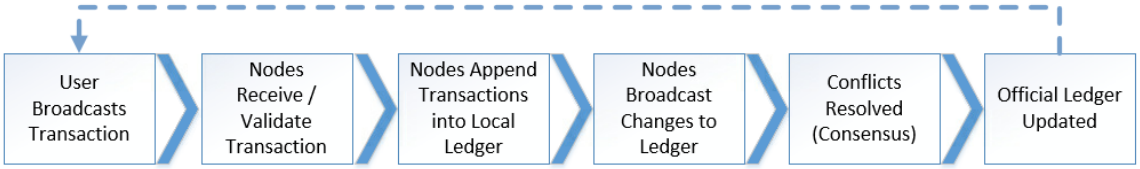

FIGURE 2.1: Generalized block production cycle for blockchain networks

Over time, the use cases for blockchain networks have expanded beyond peer-to-peer payments. With the advent of smart contracts on the Ethereum network [16], which added programmable logic that could be store and interacted with using transactions, the use cases expanded. Smart contracts enabled actions to be taken automatically when a set of conditions are met creating nearly unlimited potential use cases [16, 39, 49, 58, 60, 68]. The conditions for execution can be set through interactions that are stored as transactions sent to the network. Transactions can also act as the trigger to execute portions of the smart contract code to update the state of the contract.

The additional features enabled through smart contracts allow for complex applications to be integrated with a blockchain. A developer can create a front-end application that provides a user-friendly interface to read data from the blockchain and present it to the user. Similarly, the applications can be designed with features that allow users to send data to the blockchains via transactions. The integration of front-end user applications with a blockchain for a back-end database is commonly referred to as Web3 technology. Web3 components can be seen as a distinct layer in a blockchain architecture hierarchy. Other

components can similarly be grouped together to create a 4 layer model for blockchain architectures. A sample layered model can be seen in Figure 2.2

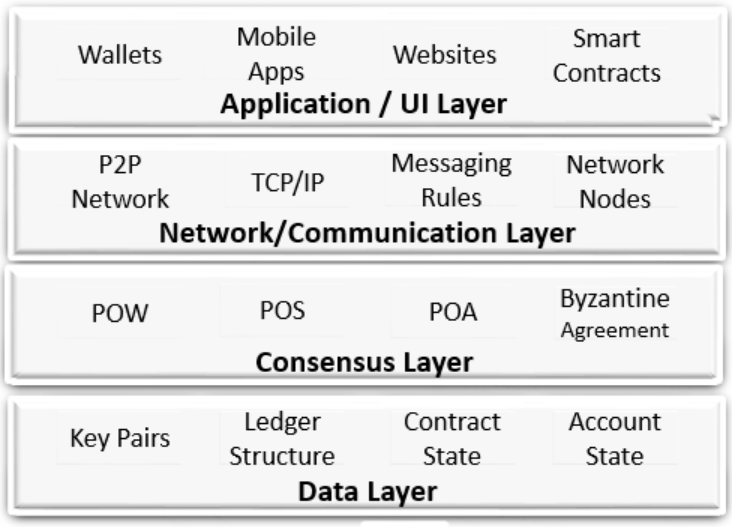

FIGURE 2.2: Sample layered model for blockchain architecture

In general, blockchain implementations can be grouped relative to who can participate in the consensus process (permissioned vs permissionless) and who is able to read the data stored within the ledger (public vs private). A permissioned blockchain means that only specified users are able to append transaction data and is similar to an immutable distributed database. Permissioned blockchain implementations have a high level of security and can operate faster due to built in trust relationships between nodes, but also often have a high level of centralization. Anyone who would like data recorded must interact with one of the nodes with appropriate permissions to do so rather then perform the action themselves. Conversely, permissionless blockchains allow any user to set up a node and participate in the validation process based on the rules and protocols of the respective network.

A public blockchain is one in which any user is able to query data from the recorded transactions regardless of who the transaction initially belonged to. Conversely, a private

TABLE 2.1: Types of blockchains

	Permissioned	**Permissionless**
Public	Hyperledger (Sawtooth) EOS	Ethereum Algorand Bitcoin Polygon FAWAC
Private	Hyperledger (Sawtooth) Hyperledger (Fabric) Ethereum	

blockchain is one in which the ability to view the data is restricted to only authorized users. Table 2.1 displays a two dimensional representation of the types of blockchains. Additionally, it indicates where some of the most popular blockchain networks fit. In the table, Ethereum is displayed twice because it is possible to implement a completely private version of Ethereum blockchain on a smaller subset of nodes by using the source code in a closed network. Similarly, Hyperledger Sawtooth [42] has the ability to be customized and configured to to operate as either a public or private, while Hyperledger Fabric [41] can only be used for private instances. A comparison of some of the most commonly used blockchain networks can be seen in Table 2.2

2.2 CONSENSUS MECHANISMS

In a blockchain, the nodes which make up the network must maintain exact copies of the data stored by all other nodes to prevent faulty behavior or attacks [10, 29, 49]. Regardless of how a blockchain network is structured, the distributed nature of the system implies that the various network nodes may at some time produce different results for the transactions

TABLE 2.2: Summary of popular blockchains

	Consensus	**Smart Contract Support**	**Public or Private**	**Permissioned or Permissionless**
Bitcoin	PoW	No	Public	Permissionless
Ethereum	PoS	Yes (EVM)	Either	Either
EOS	DPoS	Yes (EVM)	Public	Permissioned
Algorand	PoS	Yes (Non EVM)	Public	Permissionless
HyperLedger	Various	Yes (EVM)	Private	Permissioned
Polygon	PoS	Yes (Non EVM)	Public	Permissionless
FAWAC	Adversarial Consensus	Yes (Non EVM)	Public	Permissionless

included and the order in which the transactions are processed. A network consisting of many independent nodes, all with different data, is not very useful, so some mechanism must be in place to ensure all of the nodes within the network agree on a single list of transactions processed in order. Even in permissioned blockchains, wherein the nodes that are eligible to append to the ledger are known, a mechanism must be in place to make the final determination regarding which transactions are added and by which node.

Consensus mechanisms use some specific criteria to determine which node is given permission to create the next block of data. Consensus mechanisms often include some type of incentive, usually in the form of the network's respective currency, to encourage participation and to reimburse users for potential costs associated with operating a node.

Once the node is chosen, it constructs a block out of a pool of pending transactions, and the block is propagated across the network using a peer-to-peer gossip protocol. Consensus mechanisms may be different from one network to the next, but permissionless networks

tend to fall into the categories of Proof of Work (PoW), Proof of Stake (PoS) or committee-based consensus. It is also possible that some novel mechanisms include overlap across the three main areas; however, in general, they are closely related to the three main categories. Permissioned networks, where there is more control over the participation of nodes, alternative methods such as Proof of Authority (PoA) may be used. For PoA, only pre-approved and trusted nodes can submit data, allowing for simplified round-robin approaches for choosing which node is allowed to create a block. This section will cover introductory details for each of the consensus mechanisms as well as their limitations.

2.2.1 Proof Of Work (PoW) Based Consensus

The first widely accepted consensus mechanism was PoW and was a pivotal part of the Bitcoin network [59]. It is the baseline implementation from which all other PoW consensus mechanisms have been developed. Consensus mechanisms based on PoW secure the network by slowing down the process of block creation using denial of service prevention techniques originally outlined with Hashcash [8]. The PoW family of consensus mechanisms work based on an assumption that parties involved are competing for the right to append their block to the blockchain and that only one party at a time will be given that right. When using PoW, each node tries to independently solve a problem that is difficult to complete, yet easy to verify. The solution to the problem is included along with other data when creating the block providing proof that the node found an acceptable solution and has performed the required work to be chosen as the block producer.

The expenditure of resources required to solve the problem is used as a security mechanism for the network to reduce the feasibility of a malicious actor from altering the accepted history of the blockchain. The expenditure of resources imposes a cost with block creation

and as a result an attacker will need expend a greater amount of resources than other nodes in order to attack the network. In a traditional approach, a resource intensive computational problem is incorporated into the verification process prior to appending the ledger and the solution is incorporated into the metadata [5, 6, 16, 59, 73]. For the Bitcoin blockchain, the PoW implementation involves finding a nonce which when added to the other data in the header of a block and is hashed twice (using SHA-256) and must meet a specific criteria; in this case being below a target value [5, 16, 59].

The transactions included in the block are hashed together using a Merkle Tree structure, with the first transaction being the "Coinbase" which includes the reward given to the block producer when they create the block. A Merkle Tree, is data structure that can be used to create a unique hash-based digest of content, maintaining the integrity of the order of the data. Each leaf node is hashed with the direct neighbor to create a new hash value. The same approach is taken for each pair of values within a row, creating a new row of data containing the pair-wise hashes of the lower level. The process continues until only a single hash remains which acts as a integrity-preserving hash of the content of the tree. If any data or the order of the data is changed, the tree can be examined to quickly identify where the change took place. A sample Merkle Tree can be seen in Figure 2.3.

In Bitcoin, the hash root from the Merkle Tree is stored in the block header along with the previous block hash, the time stamp for when the block was created, a reference to a target difficulty, a version number and a nonce [59]. All of the values in the header are hashed together and if the hash meets the target difficulty (a numerical value the hash must be below) it is accepted. For a given block, all of the values in the header are the same with the exception of the nonce during the process of creating a block. Each node will brute force combinations of nonce values to find one that produces a hash that is below the

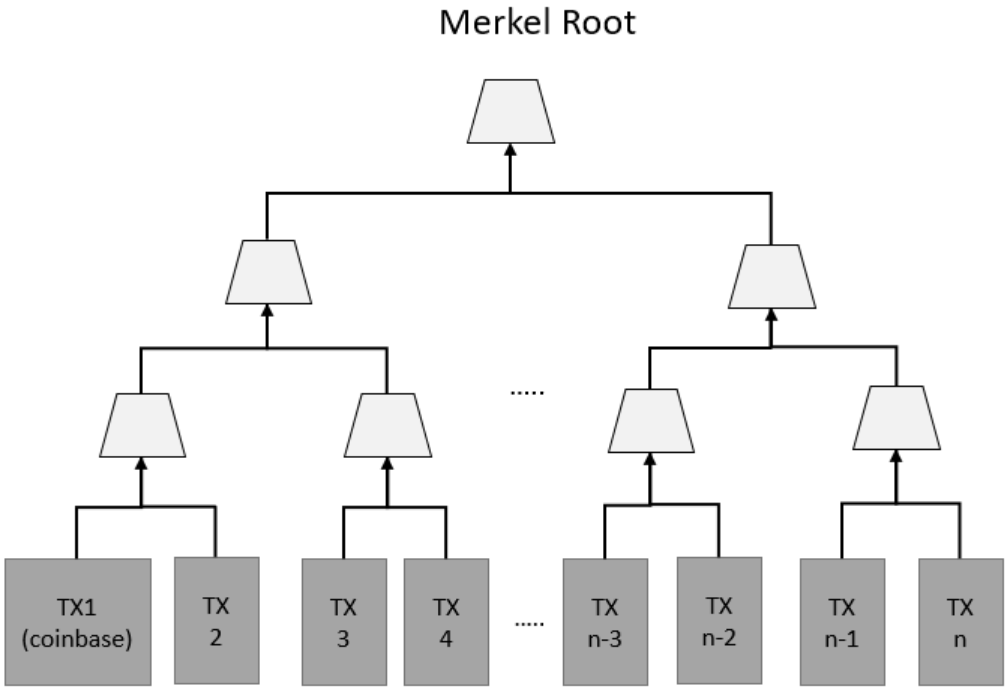

FIGURE 2.3: Sample Merkle Tree with blockchain transactions

target value. If all nonce values are used up for the size allowed, other data such as the time stamp or transaction order can be modified. The process continues until a node finds an acceptable combination of data in the header to meet the difficulty value. The difficulty value is adjusted regularly to ensure that block creation takes an average of 10 minutes. If nodes find solutions faster, the difficulty is increased, if the average is slower than 10 minutes, the difficulty is decreased. While it is difficult and resource intensive to find the combination of values that create an acceptable hash, it is easy for other nodes to verify. By hashing the values in the header together, each node should be able to verify that the values proposed by the node are acceptable.

If the PoW problem is too easy to solve, it is possible that multiple nodes create valid solutions and blocks at the same time. When this happens, it is possible for the network to

"Fork" where some nodes follow one block and other nodes follow the other. A mechanism to resolve such forks, such as longest chain acceptance need to be incorporated as well as part of the consensus mechanism to settle upon an agreed history of the blockchain. Finally, it is important to mention that significant work would be needed to create a ledger that is either longer or has a higher weight than the original ledger using this type of approach. However, such conditions are still vulnerable in situations where attacker amasses most of the computational power within the network [16, 23, 58, 63]. If a node has the majority of the computational power, the node can create blocks faster than every other node and potentially rewrite the blockchain history from some previous point in the network. The above scenario is commonly referred to as a 51% attack. The exact amount of hashing power required changes over time and is highly dependent on the number and respective hashing power of other nodes within the network [33]. For example, February 2024, the Bitcoin hash rate was approximately 540 Exahash per second meaning an attacker would need to be responsible for more than 271 Exahash per second contributing to that number[1]. An attacker with a majority of the computational hashing power would be able to pick an arbitrary block at some point in the history, make a change and rebuild the fork from that point. Eventually, the fork would catch up and surpass the honest chain since more power is being used on the malicious chain than the honest one. This would cause previous blocks to become orphaned and potentially result in double spending scenarios [23, 58].

The process of calculating the hash is fairly straight forward and as such has resulted in a type of arms race for machines that can calculate the hashes faster. While the early days of the Bitcoin network had nodes that were powered by desktop and laptop computers using traditional CPUs, eventually more powerful GPUs were used followed by specialized and powerful chips called Application Specific Integrated Circuits(ASICs) [13]. Currently,

[1] https://www.blockchain.com/charts/hash-rate

the majority of hashing power in the Bitcoin network come from large warehouses with thousands of ASIC servers, causing both the power consumption and hash rate (hashes/per second) to skyrocket. An snapshot of the Bitcoin hash rate over time can be seen in Figure 2.4.

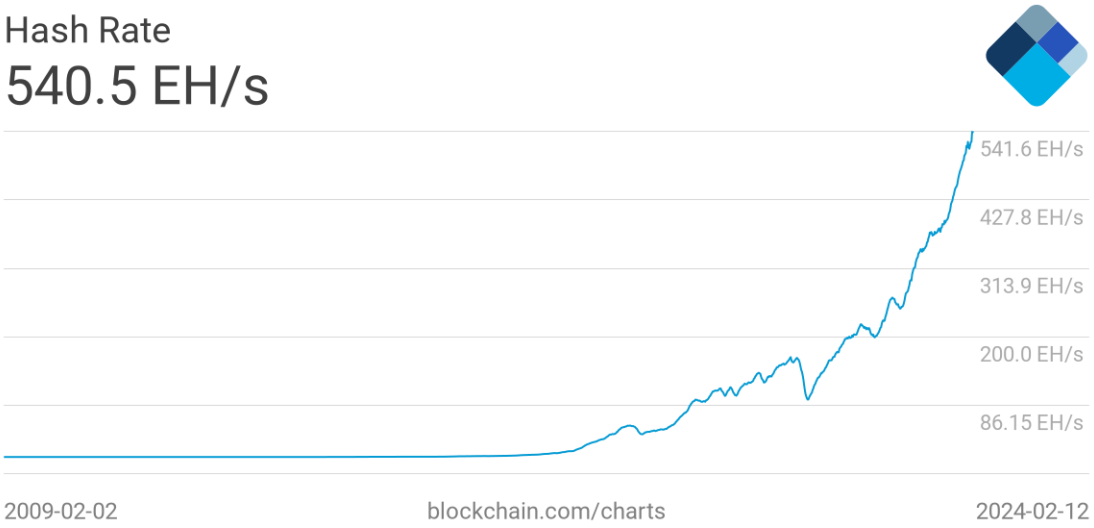

FIGURE 2.4: Bitcoin hash rate over time (2009-2024)

Other limitations of PoW implementations are related to slow transaction processing and high energy requirements associated with conducting the work [22, 23, 37]. As of February 2024, the Bitcoin blockchain uses an estimated 140.24 terawatt-hours of electricity annually, which is more than the entire country of Ukraine[2], for the simple use case of solving a hash puzzle.

When a node finds an acceptable solution, it sends the new block to other peer nodes it is aware of. The nodes that receive the block, will verify if it meets the criteria and if it does, they will propagate the block to their peers, abandon work on their current block and attempt

[2]https://digiconomist.net/bitcoin-energy-consumption

to the process to create a new block with the hash of the newly received block as the previous block hash value. It is beneficial for the nodes to abandon their work since any time spent working on the previous block is time they are not using to work towards the next block. It is possible for two blocks to be created at nearly the same time and for part of the network to expand on one of the blocks while the rest of the network expands on the other and so a mechanism is in place to handle this situation. Both blocks are technically correct and accepted, but nodes will only pay attention to the first correct block they are notified about. The fork is resolved using the longest chain rule and whichever version of the chain creates the next block, will be deemed correct and the other version will be abandoned [5, 59].

An alternative PoW implementation was previously used by Ethereum [6, 16] where instead of a computationally intense problem a memory-hard one is used. The memory-hard problem requires the node to look up data from a graph structure stored in memory to create a memory bottleneck versus a computational bottleneck. The result is that the mechanism is resistant to the use of ASICs [70] and therefore easier for an average node to participate in the validation process. Similar to Bitcoin, Ethereum defaults to the longest chain rule in the event of a temporary fork in order to achieve consensus across all network nodes. The Ethereum PoW mechanism was used from 2014 until 2022, but has now shifted to a PoS solution in order to become more energy efficient.

Other alternatives do exist as well, and while each does have slight difference in implementation, the main concept of performing arbitrary work in order to be selected remains the same. What is important to understand from PoW based consensus is the requirement to use a difficult to solve but easily verifiable problem in a competitive manner so that only one node is chosen to append the ledger. The use of one-way hashing with the solution and additional meta data secures the ledger since the work would need to be redone if a change

was made while the main ledger continues to grow. Since a single change in the transaction history would result in a completely different Merkle Tree hash, which will ultimately change the data in the header and further require a new solution to the computationally hard puzzle in order to generate a new hash that meets the validation criteria. If the malicious actor is able to solve the puzzle and create a valid block, the hash of the previous block value will no longer match the hash stored in the next block and a fork is created at that point. Most nodes will use the longest chain rule and ignore these blocks, however a malicious node can continue attempt to build on their alternate chain.

2.2.2 Proof Of Stake (PoS) Based Consensus

Although PoW has been demonstrated to be a secure method for achieving consensus, it does suffer from a few drawbacks. The first of which is that computationally hard puzzles used to secure the network also tends to be wasteful in terms of electricity or computer resources. PoS offers an alternative in which the node chosen to append to the ledger is not chosen based on the ability to solve the puzzle, but rather through the amount of currency the user has invested in the network. Most blockchains include some type of currency which can be used to fuel the network through fees or provide an incentive for node participation. PoS consensus takes into account the amount of currency that an end user has and uses it as input into a weighted random function to select the node which will append the ledger. A core concept of PoS is that the more a currency a user has invested in a network, the more likely they are going to want to ensure the network remains secure, since a failure of the system would result in a loss of value of the currency.

In PoS, the probability of being selected is proportional to the amount of stake, in the form of currency, the node has in the network. The implementation details for selecting

a user based on their respective ownership of the currency varies from system to system. In a true PoS all nodes are considered as candidates for block production and selection is proportional to the amount of stake they have in the network. However, it is also common to have an additional step or restriction applied as to the selection mechanism. For example, Ethereum's implementation of PoS requires users to manually lock their currency in a smart contract in order to be considered [17, 18]. For Ethereum, only the locked currency is considered instead of all available currency on the network. In addition, it provides a means to penalize users that misuse the system forfeiting access to the stake if the node proposes invalid block data.

Nearly all PoS based systems incorporate other elements of consensus, such as PoW and limited commitee-based consensus, creating hybrid approaches. One such example is Peercoin(PPCoin) [65], which incorporates a coin age property as an input for a function to dynamically adjust the difficulty required for a PoW puzzle. The coin age is the length of time a node has had the digital asset tied to their account. Another PoS Hybrid approach is Algorand [38] which incorporates a node's total share of the digital asset to increase the chance the node will be selected. However, rather than only a single node being chosen, multiple nodes are selected as part of a committee to perform a type of committee-based voting for validation. The Algorand selection algorithm uses a verifiable random function to determine if a node has been selected. Each node use a key and a random value to generate an output which can be used by other nodes to verify the node was selected [38]. Since no computationally intense actions are used by the nodes, the block creation process is more energy efficient and can be executed faster.

Regardless of the criteria used to select the node, once selected a private key is used to sign the submission and the information is broadcast to other nodes in the network. All

online nodes are then able to validate that the node was in fact chosen to append to the ledger in real time. In some implementations of PoS consensus it is still possible for a fork to occur, due to the distributed nature of the network. Whereas the problem is typically mitigated in PoW systems by a mechanism such using the longest chain, other approaches are needed in PoS based systems. Unlike PoW systems, where it would require significant resources to simultaneously validate multiple forks, PoS can result in minimal resource costs to the node to validate multiple forks. As a result, a node could potentially monitor for forks to occur and validate all existing forks without any penalty, thus keeping them all at the same length. While there is minimal cost for a node to validate multiple forks, there is potentially a large loss that could be incurred if the node only selects a single fork and that fork is later determined to be invalid. The negative consequences of validating the wrong chain include the loss of any rewards received on that fork when the fork is abandoned. As a result, a node could actually find it beneficial to validate all existing forks versus acting honestly. This issue is commonly referred to as the "Nothing At Stake" problem, and the PoS implementation must take this into consideration to counter the problem. In addition, PoS systems are also vulnerable to a slightly different version of the 51% attack. In this case, rather than having more than half of the total computational power of the rest of the network, the node would need to have over half of the total supply of currency within the network. Since the currency has a real-world value associated with it, accumulating enough of the network's currency to conduct an attack is difficult, and in doing so creates a risk of losing the value invested if the network fails.

2.2.3 Committee-Based Consensus

While the previous methods primarily focused on the selection of a single node per round to produce a block, it is also possible to incorporate a solution where a committee of nodes work together to produce blocks for consensus. Committee-based consensus, also sometimes called Byzantine Agreement or Byzantine Fault Tolerant(BFT) consensus, is a consensus mechanism that predates blockchain technologies and has its roots in the broader research area of distributed computing. The application of committee-based consensus to blockchain implementations has been explored [25, 38, 51, 54] as a means to reduce the waste of PoW. Committee-based consensus achieves a state of agreement among nodes through a vote-like system wherein nodes determine if the results they independently calculated match the results from other nodes in the network.

The approach takes into account that in a distributed system a potential exists for non-optimal/normal behavior to occur preventing 100% agreement at all times. Such behavior deviations could be benign in nature and can include network latency, disconnected nodes or a fault in local software. However, it is also possible that the behavior is intentional and malicious in nature. It is difficult to determine if the cause of the deviation is malicious or not and as a result any faulty behavior is treated the same. In order to achieve consensus, a two-thirds majority of participating nodes is typically required [25, 54].

The problem presented with blockchain consensus using committees is that the number of nodes must be known, which may be difficult in an open system where nodes can join or leave at will. To solve the problem of inconsistent nodes temporary or transient committee made up of a small subset of the entire network on a short term basis are used [39, 54]. However, without proper committee selection requirements in place, it is possible to create

instances where nodes are unavailable or malicious nodes are selected. Some work has been done to address this such as using a known committee size that can be transient in nature [20, 25, 38, 53, 54] or through subsets of overlapping federated committees [21]. Alternative approaches involve a structured network with the use of multi-level domains [46] or clusters [47] which control subsets of the network's activity; however, these approaches are further removed from the peer-to-peer and non-hierarchical design of traditional blockchain networks.

The typical approach of implementing committee-based consensus involves a type of temporary or transient committee made up of a small subset of the entire network. One example of a committee based approach used an election algorithm that monitored the existing committee for fault conditions [54]. Using this approach, if fault conditions are detected, a new committee is elected in an efficient manner, minimizing delay to the consensus process.

Another example of committee-based consensus can be seen with EOS blockchain [39, 48] which uses a consensus mechanism called Delegated Proof of Stake (DPoS). Although the name implies it should fall under the category of PoS consensus mechanisms, the actual validation that occurs is closer to a committee-based approach than a PoS method. In the EoS blockchain, a committee is selected through votes placed by the accounts of end users based on how much of the digital asset they own. Those votes are used to elect a committee of 21 special nodes called "Block Producers" which perform the validation on behalf of the network using a Byzantine Agreement approach [39]. Although the committee can change with new elections, the committee size is always known allowing for a block to be accepted as long as 17 of the 21 Block Producers agree. Since only a small subset of all nodes participate, it is possible that the network is vulnerable to centralization risks, and it is up

to the wider user base to monitor the activity of the nodes they vote for and change votes accordingly.

Committee-based consensus mechanisms are more energy efficient than Pow solutions, however, have additional overhead compared to PoS systems due to the communication of votes between nodes before finalizing a block. The communication complexity grows as the number of nodes in the committee grows [72]. It is often beneficial to have smaller subsets of the entire population of nodes to reduce communication overhead but this leads to concerns of how committee members are selected fairly. One of the core elements of the FAWAC architecture focuses on using a committee-based consensus mechanism with a novel approach for committee selection which will be discussed in detail in later chapters.

2.2.4 Proof of Useful Work(PoUW)

Traditional PoW approaches have thus far been proven to be secure but are difficult to scale in terms of speed, and energy efficiency. In addition to alternative consensus mechanisms that remove the work element, another area that has received attention is Proof of Useful Work (PoUW). PoUW is based on the concept of replacing the computational tasks associated with PoW with a task that provides some benefit when completed. Many such implementations integrate Artificial Intelligence (AI) or Machine Learning (ML) models into the work conducted [9, 50, 55].

Coin.AI [9] and BlockML [55] are two implementations which incorporate model training to the block creation process. In both cases transactions, nonces and hashes are used like traditional PoW consensus mechanisms. These values are then mapped to the creation

of weights for a model to be trained. The parameters are applied to train the model and validated against a set of test data. The node which produced the best results has their respective block accepted [9]. BlockML also incorporates an extra layer of complexity through the incorporation of IPFS for the storage of test data, training data and results.

The primary limitation of existing approaches is that they are limited to the specialized domain of ML model training. Further, while the approaches preserve the security benefits of PoW, they also maintain the slow transaction processing as blocks are not created until training is completed and a model which meets a specified threshold is found. Although the work accomplished is potentially beneficial, it still results in slow block production and could lead to centralization through the requirement of having specialized hardware to create the best models the fastest. Finally, the process is dependent upon a sufficient backlog of ML specific problems being submitted to the chain. If the backlog is empty, it may be possible the chain will halt until new projects are submitted.

As we will further discuss in later chapters, the FAWAC architecture solution builds upon the previous research of PoUW in two main ways. Primarily, we enable arbitrary types of code to be submitted for work and secondarily we separate the work completion from transaction validation. The result of arbitrary types of code expands the types of jobs to be used within the backlog and prevent work depletion. By separating work completion and transaction processing into two distinct portions of the network, transaction processing speed is not dependent upon the time it takes to complete the submitted work items. Work completion provides a credit for nodes which can be exchanged when the node is requests to be considered as part of the transaction validation committee. As a result, work item completion can be conducted in parallel while committee members perform fast transaction processing. Since committee sizes are much smaller than the entire node population, the

completion of a single work item could create potential candidates for several committee selection processes.

2.3 INTERPLANETARY FILE SYSTEM (IPFS)

Another core technology related to the FAWAC architecture is the Interplanetary File System(IPFS) [14]. The IPFS network presents a new paradigm for the storage and sharing of data through peer-to-peer nodes. It attempts to solve several existing problems and limitations with current file exchange protocols such has HTTP, by combining features of several other decentralized technologies into a single protocol stack to create a global distributed file system [28].

The IPFS network in a made up of a collection of independent nodes which share that has built upon existing approaches for peer-to-peer file sharing [40, 47]. Nodes participating as a part of IPFS host data that use content-based addressing. Files are stored locally and hashed to create a file fingerprint. The fingerprint has additional meta data, related to the hashing algorithm used and IPFS version data to create a Content Identifier (CID) for the file. The files are broken into 256 KB chunks of data and distributed across the network using Distributed Hash Tables (DHT) for lookup and retrieval. A example of how a file is broken into chunks, hashed and stored on the IPFS network using a DHT can be seen in Figure 2.5.

At its core, IPFS is based on pre-existing concepts utilized by technologies such as DHTs, Block Exchanges, Version Control Systems and Self-Certified File Systems(SFS)[28]. IPFS uses a variation of the S/Kademlia and Chord DHTs [11, 64]. S/Kademlia is used within IPFS for node creation and incorporates the block exchange model from BitTorrent

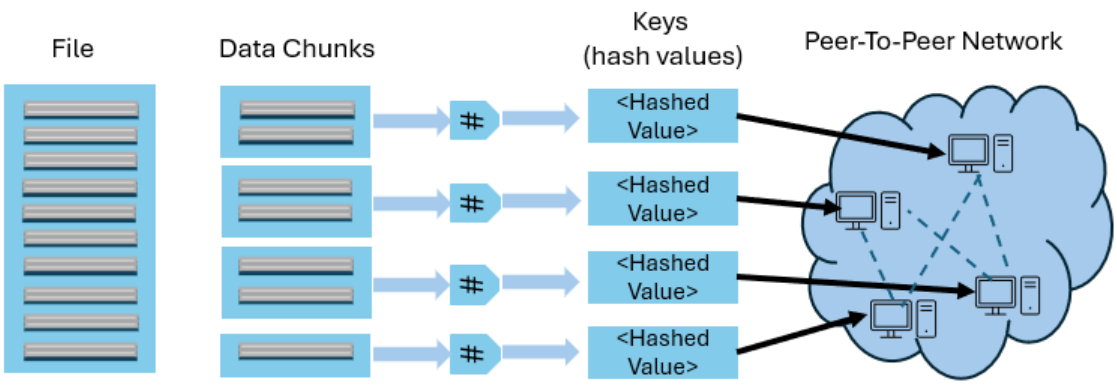

FIGURE 2.5: Example storage of a file using a Distributed Hash Table with IPFS)

to enable the lookup of nodes which host data being requested in an efficient manner. The use of an Self-Certified File System structure in IPFS allows nodes to perform remote addressing of file systems as well as provide verification of the server nodes by utilizing a hash of the public key offered by the server. Together, key components of these concepts were adapted and incorporated into the structure of IPFS to create a novel peer-to-peer distributed data storage network.

Direct communication between nodes without a central server to manage the network is one of the core fundamental principles of peer-to-peer networks. Early peer-to-peer networks lacked effective ways to organize the network to allow for easy lookup. This often resulted in flooding of requests for files to all nodes until the correct node hosting the file was found. This method can lead to excessive bandwidth use or potentially produce broadcast loops causing it to be inefficient and often results in poor load balancing [28]. Additionally, a node in the networks would host its own files within its own physical device. However over the years, many advancements in the area of peer-to-peer networks have developed and now peer-to-peer networks enable a simplified method of node/file identification through the implementation of DHTs and files could be uploaded to the network and stored on a separate

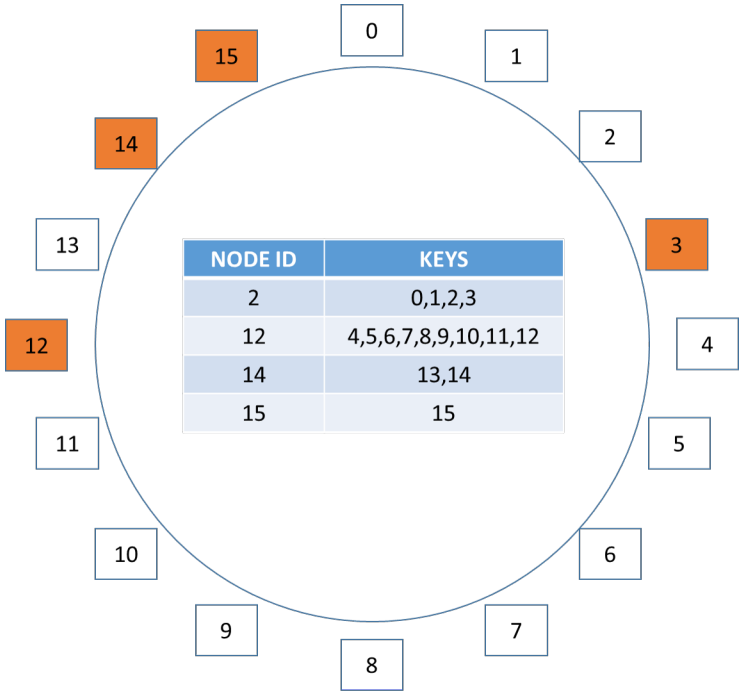

FIGURE 2.6: Simplified DHT Ring

node based on the selected DHT implementation. These DHTs contain a unique key for files stored in the network and the nodes maintain a list of the locations of the files identified by the unique key[44]. Nodes are given an ID consisting of a hash within a specified number space. Depending on the specific implementation of the DHT, IDs may be assigned randomly or may use a unique identifying factor of the node such as an IP/MAC address.

Examples of DHT implementations include Chord, Pastry, Kademlia, S/Kademlia [28]. Chord is an implementation that organizes the nodes into a virtual ring structure that uses the same number space for node and file identification. The same hashing algorithm is used for both the nodes and the files and a file is stored in the closest node that has a hash value greater than or equal to the file's hash. Figure 2.6 shows a simplified version of the Chord DHT structure with the capacity to hold 16 nodes/files. The highlights which files would be assigned to a respective node using the approach previously introduced.

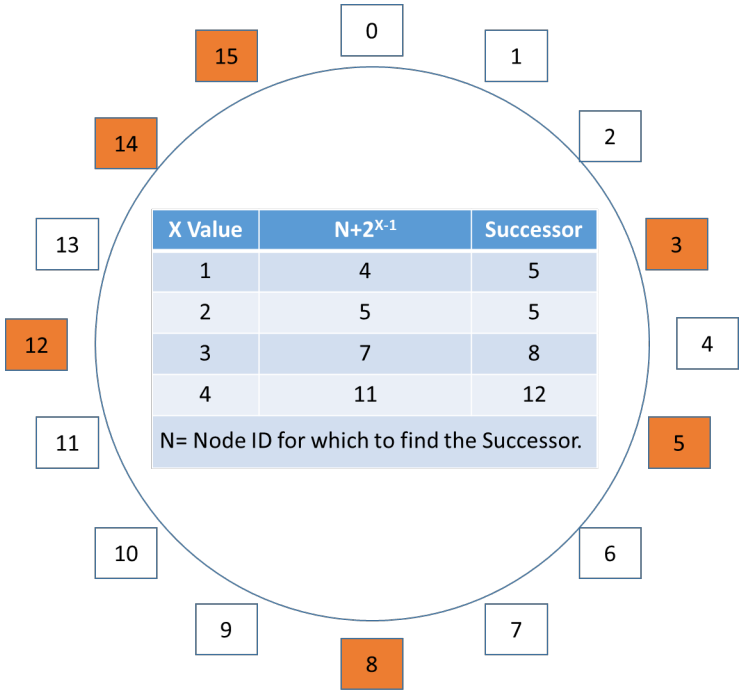

FIGURE 2.7: Successor Nodes for Node 3

Chord allows for easy lookup of nodes by creating a routing table which only contains a small subset of the entire ID number space. Chord identifies successor nodes by applying a formula for N additional nodes, where N is the number of bytes used to make up the number space. Figure 2.7 outlines a simplified example of how successor nodes for node 3 are calculated. Because there are 16 nodes in the example figure, only 4 entries are used in the table because 16 can be represented in binary with 4 bits. For each Bit possible (1-4) the value is calculated by adding increasing powers of 2 to the node's id. Distance for nodes in Chord are calculated by traveling around the ring clockwise meaning the distance between node 3 and 5 is not the same as the distance between node 5 and 3. It is also important to note that the term distance does not account for geographic location or separation, but rather how far away the hash of one node is to that of another.

Another popular DHT, Kademlia, is widely used due to its simplicity and low overhead.

Kademila expands upon Chord by using an XOR comparison between nodes to determine distance. When using XOR in this way, the distance between node 3 and 5 would be the same as between 5 and 3. Kademila is the DHT used by the popular P2P application BitTorrent. S/Kademlia is another DHT that expands upon S/Kademlia by introducing functionality that provides protection against Sybil attacks by creating a PKI key pairs during the node creation process. A Sybil attacker can create multiple malicious accounts or nodes to produce a higher reputation for their malicious nodes, thus attracting honest users to their systems.

The design of IPFS is made up of a synthesis of the previously discussed technologies in order to create a stack of seven sub-protocols integrated together as part of the overall IPFS protocol [28]. The sub-protocols that make up IPFS are Identity, Network, Routing, Exchange, Objects, Files and Naming. Identity generation and verification is handled through an implementation of the S/Kademila mentioned earlier. Each Node has an ID, a public key and a private key. S/Kademila is used to generate the public/private key pair and the ID is determined by hashing the public key of the node. When establishing a connection, the public key is sent and if the hash of the ID does not match the public key the connection is aborted.

Routing is accomplished by creating and updating a DHT based on S/Kademila but also takes concepts from an alternative DHT called Coral. For small files less than 1KB, the data is stored directly on the DHT, however larger files only store a reference consisting of a Hash of the contents. Once routing of files is handled, Block Exchange occurs using a protocol called BitSwap. BitSwap operates similarly to BitTorrent but also adds the ability to monitor and track a nodes contribution by tracking bytes sent versus bytes received. An excess of bytes sent will lead to a positive credit which could be used to infer that the node is not malicious and favor that nodes use over nodes with a low or negative credit. This credit

is partially implemented through the use of a Ledger which keeps track of bytes sent and received between two nodes. This ledger can be used to determine if a peer is trust worthy or if it should be ignored.

Object handling makes up the next sub protocol of IPFS and hare handled by a Merkle DAG. Objects implemented through a Merkel DAG construct can be used for content addressing, provide tamper resistance and enable de-duplication as all objects that have the same content are only stored once. As IPFS acts as a global file system, objects are able to be traversed with using paths made up of the hash of the object and the name/path to the object. Additionally IPFS provides the capability for Objects to be protected through encryption which also protects access to links to the file by making it impossible to navigate to the file without the decryption key.

Files make up the next part of the IPFS protocol. Files can exist either as Binary Large Object (blob), lists, trees, or commits. This model is similar in construct to what is used within Git. A blob contains an addressable unit of data and represents a file. Lists can be made up of a large blob or several smaller blobs stored in a sequence. It is possible for a List to also contain other lists. The links themselves, contain the hash of the object stored and a reference to the size of that object. A tree is similar to lists except they also include another stored field to represent the name of the object stored in the tree. The hashes within a tree can represent blobs, lists, other trees or commits. Finally a commit is a snapshot in history of an object. It includes a reference to the type of file object, the date of the snapshot and a message associated with the commit in addition to the file as it existed at the time of the snapshot.

The final piece of the IPFS protocol is the Naming sub protocol referred to as Inter Planetary Naming System (IPNS). IPNS provides an ability to provide mutable naming for

content so that it can be more user friendly and human readable. SFS is used in order to provide users with a mutable name space for a node at /ipns/<NodeId> this is contrary to /ipfs/<NodeId> which contains a reference to immutable date stored in IPFS. This can be expanded to be even more user friendly by using a DNS TXT record to reference the hash of a value as a human readable name.

A system running an IPFS node, has the ability to upload files that are stored locally into a partition that is accessible to the rest of the IPFS network. The uploaded files are hashed to create a Content Identifier (CID) and and made available to other IPFS nodes that know the respective CID. When another node accesses a file stored on IPFS, a local copy is created on the node and the node begins hosting the file for other peers to find as well. Not every node is connected to every other node, however through chained look-ups in the hosted DHTs referencing node IDs, a node that is hosting the data will eventually be found. If the number of nodes hosting a file is low, the process of lookup a file can be slow and prone to delays. However, if a file is present on many nodes, the lookup is quick and can be retrieved in parallel. IPFS does contain mechanisms to actively update peers if the peer address is known, and this method can be used to increase file retrieval times.

Blockchain implementations have previously used IPFS integration to store static data off chain for smart contracts or as part of a blockchain incentive for distributed file storage [6, 16] Regardless of the size of the file, a relatively small CID is all that is needed to retrieve the file. Smart contracts can reference data stored on IPFS using the CID, which can be integrated with a front-end framework to retrieve the CID from the blockchain and render it to the user. While static data storage makes up a major part of IPFS-Blockchain integration, Filecoin [15] is another project that was created that uses IPFS as a core component. Filecoin acts as a distributed storage service, where end users run an IPFS node

and a Filecoin server. End users place offers for users running Filecoin nodes to store data for them. As the Filecoin nodes store the files, checks occur to verify that the file is still present and reward the nodes for successful checks.

The FAWAC architecture has a deep integration with IPFS in order to efficiently exchange data between the various enclaves of the framework, and ensure data integrity. It further expands upon IPFS capabilities by demonstrating the that arbitrary code can not only be reliably retrieved and executed, but that the content-based addressing of IPFS acts a reliable indicator for work execution, an important part of the advanced work portion of the architecture. The specifics of how IPFS is used will be covered in more depth in future chapters.

2.4 VERIFIABLE COMPUTATION

In addition to operating as a blockchain to process transactions and using IPFS for distributed data storage, the FAWAC architecture also contains a mechanism for remote code execution for useful work. Part of the criteria for committee member selection within the framework involves nodes executing computationally intensive jobs submitted by users. In order to verify that a node successfully completed the jobs and get proper credit, a mechanism must be in place to perform the validation. Therefore, in this section we provide a brief overview of the concept of verifiable computation before discussing the specific mechanisms used by FAWAC in a later chapter.

A popular solution for execution of long running or computationally heavy programs is to offload the code to be executed by external nodes. Cloud services offer a centralized solution but may have additional security considerations and financial considerations [31].

Offloading computations further away from cloud service providers towards end devices, a concept called Fog computing has recently gained popularity. Fog-based solutions rely on a connection to a centralized cloud provider for coordination and are thus not true decentralized approaches.

A more decentralized solution is grid computing [34] but has its own difficulties related to information dissemination retrieval and scalability [19]. Each approach has security issues to overcome to ensure the safety of the remote device executing the code. The problem of executing malicious code can be addressed through the isolation and restrictions that can be applied to container-based environments coupled with trusted execution software which limits the potential for damage to the host machine [12]. Additionally, hardware-based solutions such as Intel's SGX technology have been used protection of end user devices during execution in distributed environments [69]. Our approach attempts to solve the problem without the use of specialized hardware or container software required on the end node for protection. Rather, the approach used relies on the analysis of code written in interpretive programming languages to prevent the potential for malicious activities from occurring.

Ensuring that the code is safe to execute by distributed nodes is only one part of the problem we attempt to solve which is also examined by other researchers. Another component is related to ensuring that the executed code generates correct results. One approach is to incorporate verification that can be analyzed based on the compilation process and verification key such as with Pinocchio [61], Geppetto [24] or PinochioQ [35]. Implementations using a variation of the Pinocchio compiler create a cryptographic key used as proof as part of the compilation process.

Alternative verification methods include architecture that establishes mutual trust between nodes [66] or graph-based social structures [56] providing an incentive model for

providing correct computations. The primary difference between existing techniques and our approach is that we outsource the verification process to a third-party Coordinator who determines validity through comparison of multiple result submissions.

The existing works examined here provide great solutions to specific aspects of the problems we are addressing. However, the works focus on either security or code validation independently. Conversely, our approach attempts to tackle both areas simultaneously by combining related topics into a unified solution. Further, the FAWAC solution does not require participants to rely on specialized hardware or software to provide both security and verification. Rather we use basic concepts of the Python programming language which is widely available and highly readable, and the hash-based addressing of content used with IPFS.

CHAPTER 3

FEDERATED ADVANCED WORK ADVERSARIAL CONSENSUS (FAWAC)

In the preceding chapters, we provided a foundation of concepts related to existing technologies to include blockchain, peer-to-peer networks, IPFS, consensus algorithms and verifiable communication. Each of these technologies act as a core component for the FAWAC architecture. To understand how each component fits into the FAWAC construct, it is best to briefly provide a high level look at the end state and work backwards to see how each part plays a role. Each of the individual components will be discussed in greater detail in subsequent chapters.

The overall goal of the FAWAC architecture is to create a secure blockchain that uses a committee-based consensus mechanism incorporating a reputation system for committee member selection. Committee-based consensus has the benefit of being faster and more energy efficient than PoW systems, but its security is related to the network's ability to reliably select honest nodes over malicious ones. Working backwards, a reliable reputation system is something that needs to be designed. The FAWAC architecture utilizes a reputation system based on two primary considerations. The first is based on past performance as a validation node, such that previously identified malicious activity penalizes the node and honest activity rewards the node. To build a reputation system, a mechanism needed to

be developed to record previous activity and thus the multi-block concept was developed, wherein a traditional block is split into a data block, validation block and meta data block. Together, each of the block types act as what is traditionally viewed as a single block in other block chains. However, by breaking the blocks apart, we are able to capture votes and prevent state updates when the block is rejected by only updating meta blocks if the data is accepted via the validation block. In addition, we further address the risk of nodes voting to accept without validation through the "Adversarial Consensus" component of FAWAC. Adversarial Consensus allows validation node to purposefully present false data to detect if nodes are approving nodes without actually validating first. The second part of the committee selection system aims to limit the number of malicious nodes present in the entire population as a whole. While the reputation-based system helps identify malicious activity and reduce the selection rate of those nodes over time, by itself it is still susceptible to Sybil attacks if a malicious user creates multiple nodes. To address the Sybil attack risk, we created incorporated the "Advanced Work" component of FAWAC.

The concept of advanced work is based on the idea that in order to reduce the number of malicious nodes present, some cost must be applied to the process making it impractical for malicious actors to create multiple nodes. The concept is what makes PoW and PoS systems secure. In order to be selected, nodes must have a significant share of whatever criteria is used for selection. For PoW, this can be hashing power and for PoS it could be the amount of stake controlled by the node. Splitting the power between multiple nodes effectively weakens the selection rate compared to centralizing it into a single node controlled by the malicious actor. In particular, the advanced work concept borrows concepts of PoW by requiring nodes to first complete some work to be eligible to be considered for the committee

member selection process. However, as discussed in previous chapters, traditional methods of PoW tend to waste considerable amount of energy. The FAWAC architecture mitigates this concern through the integration of useful work as the type of work considered in the advanced work component. Using a system that allows users to submit code for execution, the code is selected as a job and participating nodes complete the work and return the results. The results are stored on IPFS creating a unique fingerprint for the solution, and any node which produces the same solution will in turn produce the same IPFS CID when it stores the file on IPFS. The system incorporates validations from multiple nodes producing the same result as the criteria for job acceptance. Since the work can include execution of programs with real-world implications, the concerns about waste is minimized.

The "Federated" component of the FAWAC architecture refers to the idea that each of the supporting elements, or enclaves, operate independent from each other, yet are still integrated as a single system connected with a common IPFS back-end. Specifically, the areas of responsibility are broken into three enclaves. The three enclaves that make up the FAWAC architecture include the Work Generation Enclave (WGE), Work Coordination Enclave (WCE) and Transaction Enclave (TE). Each enclave consists of nodes working together for a common purpose despite not being controlled by and single centralized entity to create the FAWAC architecture.

3.1 FAWAC ARCHITECTURE

The FAWAC architecture includes three enclaves that work together to provide a secure committee-based blockchain system. The WGE consists of nodes, known as Work Item

Generators (WIGs) which are responsible for the creation or arbitrary jobs by end users. The jobs should consist of the type of work which an end user may want to have outsourced. The reason for which the code is outsourced is not necessarily a focus or concern for that FAWAC addresses, other than to provide a resource for outsourced computing that is reliable and cheap for end users. Currently, if a user has code they want to execute on an external system, they either need to rent space on a cloud based system or apply for time-shared access for clustered computing environments. The WGE component of FAWAC provides users with an alternative method, allowing the end users to create jobs, called work items, and submit them to the network for execution.

Work items created by nodes in the WGE are written in Python and stored on IPFS. When the files are stored on IPFS a reference to the work items is sent to nodes in the WCE. The work item is treated as a pending transaction within the WCE and is eventually selected to be worked on by a Coordinator Node within the WCE. Once selected, it is advertised and Work Item Executors (WIEs) will download the files locally and execute the code until a solution is found. When a solution is found, the results are saved to IPFS and the reference to the file is sent back to the coordinator. The solution that is stored on IPFS will create a CID that is created based on the content of the file. As such, any node that creates the work and generates an identical solution will have an identical CID associated with it. Once a sufficient number of identical solutions are received, nodes which produced a correct solution are rewarded with a type of currency called we call a credit.

Credits are used in a modified version of PoS as input for a selection algorithm which determines which nodes act as committee members to validate transactions on the TE. When a node submits a request for consideration as a committee member, the credits earned by

the node are consumed. Credits are non-nontransferable between nodes and therefore the only way to earn the credits is through the completion of work items within the WCE. It is important to note that when a node consumes the credits to be considered for the committee, they are not guaranteed a spot right away as an additional selection mechanism is used for committee member selection within the TE. The selection mechanism takes into account the reputation score from previous activity when selected as a committee member in the TE. The double selection approach helps to minimize the presence of malicious nodes within committees and thus ensuring the safety of the network. While each enclave has their own respective area of responsibility for the network as a whole, the interconnected nature is what gives the FAWAC architecture its strength. The interconnected relationship between the three enclaves can be seen in Figure 3.1.

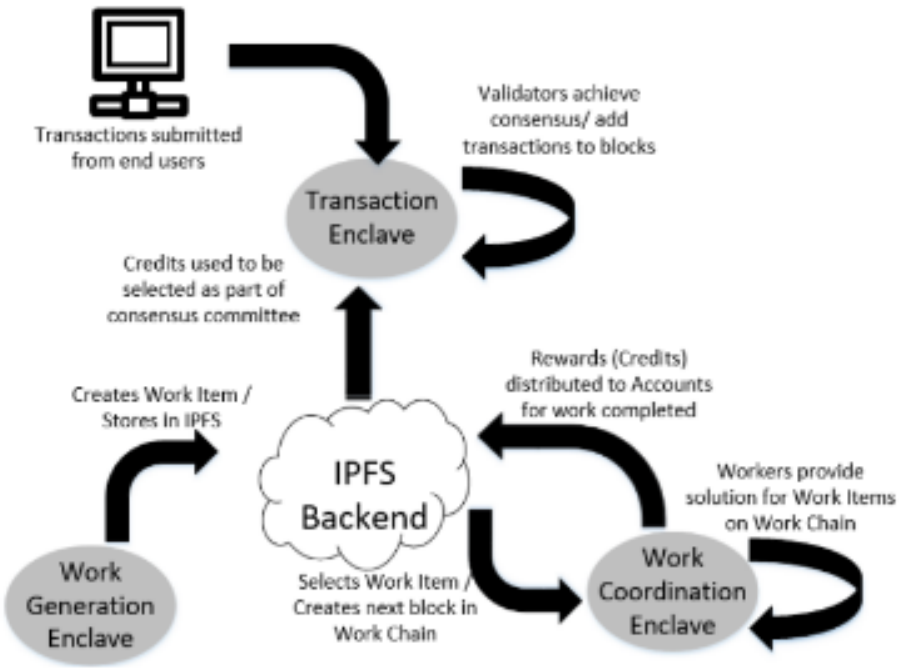

FIGURE 3.1: Interaction between the three FAWAC enclaves

3.2 IPFS INTEGRATION

We have mentioned several times in this work already that integration with IPFS is a core part of the FAWAC framework. One of the primary reasons for the importance of this integration is that each enclave operates independently of the others, however the input and output from the enclaves may be used in the other enclaves and thus a simple way to share the data is needed. In addition to running the required software to participate as part of the blockchain, each node is also be required to run an IPFS node. All data that is recorded by the blockchain resides on IPFS. Examples of items stored on IPFS include transaction data, block data, account states, reputation states, smart contract data states, public keys, work item related data, and smart contract code. Since each node is connected to IPFS, the availability of data sources for retrieval should increase as the number of nodes participating increases and as a result the speed for data retrieval should decrease.

As of June 2023, the number of IPFS nodes was estimated to be close to 500,000. [1] By connecting to the existing IPFS network, data storage can also be expanded beyond nodes participating directly in the FAWAC network. Any user that requests data from the blockchain, potentially expands the data storage even more, creating a more resilient ecosystem for data storage. The use of IPFS also creates an additional side effect due to the content vs location based addressing of IPFS. As long as a single node has a copy of the data, the data can be retrieved. This allows some nodes to potentially prune data and reduce the storage requirements for individual nodes. By design, the blocks created in the TE, limit data to references of other data stored in IPFS, effectively creating a block that consists of

[1] https://banklesspublishing.com/ipfs-powers-the-decentralized-web/

CID references. When a node validates the block data, the CIDs are extracted as needed to recreate the data and perform validation. However, if a node only stores the block data itself, without extracting the the storage requirement is minimized.

It could be argued that allowing nodes to keep the minimum data of the creates a risk that data will not be available when needed, however, this concern is addressed by through the incentive mechanism for the network. When a node is selected to act as a block proposer as part of the validation committee and creates a block that is accepted, a reward is given in the form of the network's native currency. If the block containing the data disappears, so does the reference to the reward given to the node. Therefore the node has an incentive to keep the block data where they received their reward. Additionally, since the validation of the block is performed by a committee, all members on the committee also have an incentive to preserve the data as well, reducing the risk of all nodes purging the data. It is also possible for nodes to set up long term storage separate from the node itself to store IPFS data by establishing a direct connection to the node hosting the data through peer bootstrapping.

3.3 NODE CONFIGURATIONS

The FAWAC architecture consists of nodes which can operate within each of the three enclaves at various parts of the life cycle. Since nodes exist within each enclave and share a common IPFS connection, data is able to flow freely between the enclaves. Nodes switch between operating in each of the three enclaves through configurations settings for the nodes. A FAWAC node supports 6 different components which can be enabled depending on the role the node is performing within the network. The node components include:

1. Work Generation Component (WGC)
2. Worker Component (WC)
3. Coordinator Component (CC)
4. Storage Component (SC)
5. Validation Component
6. Account

Figure 3.2 provides a comparison of several possible configurations with respect to active components.

If a node participate in a specific layer then specific components are required to be enabled. Similarly, if a node chooses not to participate in a specific activity, then some of the components can be turned off. The Account is the only component that is required for all configurations. The Account is responsible for keeping track of all forms of currency and credits owned by the node, as well as maintaining keys for all signing activities. The keys used by Accounts will use quantum resistant means such as the Dilithium Algorithm [32].

It is possible to have a node with only the Account component enabled. In such a configuration, the node can send and receive the network's digital currency or interact with smart contracts via transactions that are tracked in the Transaction Enclave portion of the network. An interactive application, called a wallet, can be used by the user to provide access to the Account without requiring the rest of the components. Further, when a node is rewarded for participation in one or more layers of the network, the rewarded currency

is also tracked using the Account. Each node which operates in the different enclaves acts as a bridge between the independently maintained ledger enclaves. Further due to a shared IPFS back-end used by each node, information sharing regarding Account balances should be achievable without discrepancy.

The Storage Component (SC) is the next most common component among possible configurations. The SC involves running an IPFS instance on the host node. Within the network, IPFS is used to store the state of transactions, accounts, smart contracts, blocks and Work Items acted on by the network. Each node with the SC enabled will be responsible for storing data related to their direct participation.

The Work Generation Component (WGC) is used to participate as one of the networks Work Generation Nodes (WGNs). The WGNs are responsible for creating Work Items that are worked on by Workers which consist of the executable code and related data, on an IPFS instance. While any node could enable the WGC, the requirement to have the component enabled allows for better control of Work Item submissions to the network. For a node to run as part of the WGE, WGC must be enabled for that node.

The Worker Component (WC) is used by nodes acting as Work Item Executors within the WCE and handle all actions related to solving the Work Items submitted by the network's WGNs. When Work Items are submitted, a node that has the WC enabled will download the code and run it locally on their respective machine until a solution is discovered or computation completes. The node will then submit the result to a node running the Coordination Component (CC) and acting as the current Coordinator.

The Coordination Component (CC) also is used within the WCE and is used to maintain control and order of the Work Items submitted via the WGNs. In order to prevent a free-for-all style of Work Item execution, nodes will be selected to act as a coordinator for an epoch within the Work Item Coordination Enclave. The CC nodes are responsible for selecting which Work Item will be worked on next, determining the correctness of the solutions and allocating the respective reward credits for correct solutions. Since the node is not able to act as a worker while selected as a coordinator, a small portion of the reward will be given as a reward for the node's involvement in the process.

The Validation Component (VC) is the final component that is possible to be enabled for a participating node. When this component is enabled, the node can participate in the consensus process of the TE. It is important to note, however, that a node cannot just enable this component and expect to participate in the TE consensus. In order to participate in the TE, the node must consume Credits which can only be earned by participating as a Worker or Coordinator in the WCE of the architecture. For this reason, a node that operates as with the VC will also have the CC and WC components enabled as well.

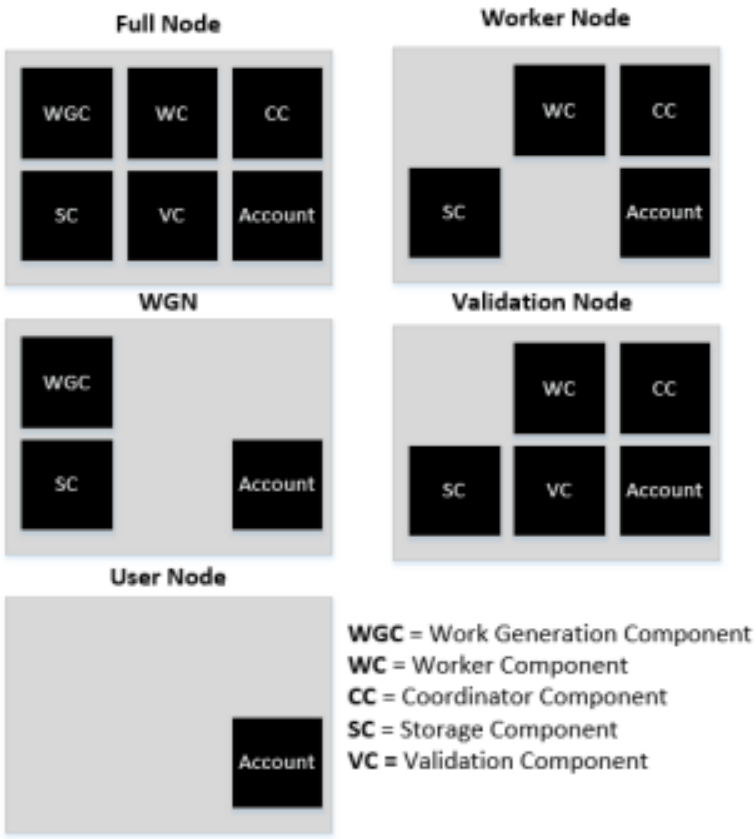

FIGURE 3.2: FAWAC node configurations

CHAPTER 4

DISTRIBUTED WORK GENERATION & EXECUTION

In this chapter we focus on the interconnection between the Work Generation Enclave (WGE) and Work Coordination Enclave (WCE) components of the FAWAC architecture. The interaction between these two enclaves make up that "Advanced Work" aspect of the FAWAC architecture. Here we will provide a deep dive into how the individual actors within the enclaves operate and interact with each other. Specifically the focus will be on how the actors generate work items, select work items, complete work items and verify results. As part of our analysis, a framework and prototype was created to demonstrate the ability to generate generic work items which are stored on IPFS, and use remote nodes to reconstruct and execute the work items.

As was discussed briefly in the previous chapter, the work items act as a mechanism to counter Sybil attacks against the network by imposing a computational cost on participating nodes. If a malicious actor attempts to create additional nodes to increase the chances of being selected for a committee, each of those nodes must also complete work items before they can be considered. While, the design could incorporate trivial problems to solve, similar to the efforts to find a valid nonce in Bitcoin style PoW consensus [59], we aim to reduce waste by utilizing the nodes to execute useful work items. Useful work is a generalized term for any type of arbitrary code submitted to the network. Candidates for the type of work can

include long-running scientific calculations, machine learning model training, or data analysis. Ideally, the network would provide a free alternative to outsource data processing for users, providing a benefit to end users while simultaneously helping to secure the network against Sybil attacks.

4.1 SECURITY CONSIDERATIONS

In order for the execution of work items to provide value to the overall architecture, two main security considerations need to be addressed. First, there is a risk that the actors creating the work items will create and submit malicious or harmful code to the network. As such, there needs to be a mechanism to ensure that the execution of code will not harm the node running the code, or be used as a launching point for a network-based attack. Secondly, there is a risk that participating nodes will attempt to produce inaccurate or fraudulent results without actually executing the code. For this reason, there also needs to be a mechanism in place to ensure the work was actually completed and to verify the results are correct, without a validation node having to execute the code themselves.

The execution of arbitrary code created by an untrusted user has the potential to create security risks for the executing node. However, the framework attempts to take this into account and incorporates countermeasures to minimize the risk. While, we describe the code as arbitrary, it is not completely the case. Code that is submitted must pass a validation check prior to being executed. All of the code is written in the Python programming language, meaning the code submitted is not compiled and is human readable. While we don't expect to have humans read and validate the code, the fact that it is human readable does

allow the use of a simple parser to scan the code for key words that could be associated with misuse in an untrusted environment. Examples include references to modules that enable network communication such as the socket, urllib, http, ftplib, and other similar modules. Other examples that could impact the operating system of the host computer include the os module, which allows the code to read file system data or even delete files. A configuration file is created with a list of restricted key words that is used by a parser. If the parser finds one of the restricted key words, it will cause an exception and abandon execution. Some work items will require additional data to be used as source data to be processed. To handle this requirement, the framework stores required data in a specified folder and analyzes any file read operations to verify the matches the specified folder.

The other security consideration is related to preventing nodes from claiming they completed the work and submitting false results. The framework also considers the verification of code output when executed by untrusted nodes in several ways. Primarily, through the use of IPFS for the storage of the results. For all nodes that independently execute the code, if they produce the same result, the same IPFS CID will be generated when the file is stored. There is no way to predict what the CID will be before submitting it, so essentially the only way to generate the correct CID is to perform the work. It is possible that malicious nodes may attempt to submit invalid results in an attempt to be gain credit, so a mechanism is needed to distinguish between which results are likely to be honest versus ones likely to be dishonest. The mechanism used involves a requirement for multiple nodes to provide identical results. The Work Item Generators can specify the minimum number of validations that need to be identical before a result is considered valid. However, this approach alone does not prevent a malicious nodes working together from sharing results. A malicious

node could potentially solve the work item and share the results with other malicious nodes. We counter this type of attack by also requiring the worker nodes to communicate with a coordination node when they start and complete the work, creating unique timestamps for their effort. The Work Item Generator can specify a minimum and maximum time range that is acceptable, which will be unknown to the workers, and therefor may not be able to predict an appropriate time range even if they have the correct CID for the result. As a final effort to minimize the impact of invalid results, the code submitted includes a built-in logging component that tracks method execution and data parameters. Python annotations, which interact with the logging function, are added to all functions when they are defined. Both the submission file and the log file are submitted when the work is complete. While the result files should be identical for all users, the log file should follow the same format but could have different timestamp values, but could still be compared line by line for validation. Even if malicious nodes are working together to share results, the only way that one of the nodes can submit a correct answer is if at least one of them actually performs the work. In this case, even if nodes are attempting to operate in a malicious manner, they are still bringing limited value to the network by contributing to the honest completion of work items. As a result, attempts to gain a malicious advantage will be balanced by providing positive results to the rest of the ecosystem.

4.2 SYSTEM COMPONENTS

The advanced work framework consists of three types of nodes responsible for their own respective actions within the network. The three nodes are Work Item Generators, Coordinators and Work Item Executors. Work Item Generators exist within the Work Generation

Enclave while Coordinators and Work Item Executors exist within the Work Coordination Enclave. A parser program is used by each of the nodes to validate that code submitted is safe and meets required standards for logging. In addition to checking if the code includes commands which are flagged as potentially harmful, the parser will also ensure each method contains a custom annotation used by the framework for log generation. Each node also runs an IPFS instance to enable the upload and download of files associated with the work items created.

4.2.1 Work Item Generators

Work Item Generators act as the starting point in the process and are responsible for the creation of code which conforms to standards and requirements needed to pass a validation check from the parser program. All programs are written in Python to enable easy analysis of the code. Python is an interpreted programming language, meaning that it is not compiled and prior to execution. Since the code is not compiled, it can be check against commands or libraries which could be harmful if executed. If the validation from the parser passes, the framework saves all required files to IPFS and generates a configuration file containing all the required information to recreate the code. The configuration file contains a few required fields and is stored in JSON format. The random seed is a value that is used to ensure code is deterministic. Some programs may require the use of random values as part of the logic. The random_seed ensures all nodes executing the code will produce the same random values for any random functions. The num_vals field identifies how many identical result files are needed in order to determine the result is correct. The number is dependent upon the requirements of the Work Item Generator but should be at least 10. Min_execution is the

minimum expected execution time (in seconds) that would be acceptable as a reasonable solution. This parameter is added as a security mechanism to discourage cheating by passing completed results to peers to submit. Similarly, max_execution represents the longest execution time that would be acceptable. The purpose is to prevent the network from stalling due to some type of faulty code. The Coordinator Node will monitor start times for all Work Item Executors and if nodes exceed the max_execution value, the work item can be aborted and another one can be chosen. The require field is used to reference the CID for the requirements.txt file for the project. The file is uploaded to IPFS and the CID is stored as the value for this field. The execute field is used to to store the CID for the code that will be executed by the Work Item Executors. The config field includes parameters that are passed as arguments at execution time. Finally the input_files field acts as a list holding both the data that should be stored, as well as what the respective file name for the file should be. The name field is required since the file name will be be hard-coded into the program itself and must match when stored on the remote machine. The configuration file is then stored in IPFS to generate a CID and the CID value is sent to the blockchain existing within the Work Coordination Enclave. A sample configuration file generated by the Work Item Generators is seen in Figure 4.1. Note that the IPFS CIDs have been truncated to save conserve space.

4.2.2 Coordinator Node

At any given time, there may be multiple Work Item Generators submitting work items to the network. Since the execution of the items can take a significant amount of time, it is possible that a backlog will be created. While a free-for-all approach, where all work items are immediately advertised and each Work Item Executor chooses their own items

```
{
  "random_seed": 42,
   "num_vals" : 10,
   "min_execution" : 3600,
   "max_execution" : 21600,
   "require": "QmVsDwUchRd...7za1sTiSVntry574dr",
   "execute":"Qmdvq2wfDtw...RL44mxDd6NgWHpUzW",
   "config":"QmbYFsWPGnmv...siJnfsh3T3tWBUiBT",
   "input_files":
   [
     {
       "name": "testData.csv",
       "content":"QmTU5MCwm...bvR2NuwxYZ1quemLE5"
     }
   ]
}
```


FIGURE 4.1: A sample config file produced by Work Item Generators

could result in parallel execution, the unordered approach could lead to longer times until multiple validations are reached. Therefore, a mechanism to choose which specific item is being worked at a given time is needed. This framework makes use of Coordinator Nodes for this purpose. Coordinator Nodes are responsible for selecting which work items submitted by Work Item Generators will be worked on next. By selecting a single item for all nodes to work on, the framework reduces complexity of tracking the status of multiple work items which could be in various stages and speeds up the process of receiving multiple results needed for validation. The Coordinator Node also acts as a centralized source for tracking execution times and collecting results for validation. When a work item is selected by a Coordinator Node, it will create a block on the blockchain which includes a reference to the work item CID. The Coordinator Node will then propagate the message to other network

nodes using a gossip protocol. If for some reason, a Work Item Executor does not know what the current work item is, the Work Item Executor can also query the Coordinator Node directly to receive the information.

When a work item has been selected by the Coordinator Node, the Work Item Executors can submit a message to the Coordinator requesting to start working. The Coordinator Node will keep track of which Work Item Executors requested to start working and at what time in a local ledger. After a Work Item Executor complete the work, they also send a message and another entry is made. The Coordinator Node uses the two timestamps to determine the execution time and evaluates if it is within the parameters in the configuration file. It is important to reemphasize that the minimum and maximum values for execution are not shared with Work Item Executors so the only way to receive credit is to actually perform the work which, if performed honestly, should fall within the acceptable window. Further, each Work Item Executor will also submit their own respective log file along with the result. While the order of data produced by the log file will be identical across nodes, timestamps and references to the executing node will be different resulting distinct CIDs. The Coordinator Node will download each of the log files and validate it meets the expected fingerprint and if so will be accepted.

Once a result is accepted, the CIDs for the result and the log file are submitted back to the Work Item Generator and Work Item Executors which successfully completed the work are rewarded by recording the rewards in a new block on the blockchain. The Coordinator also selects the next node to act as a coordinator and stores this information in the new block along with the rewards for participating Work Item Executors.

4.2.3 Work Item Executor

Work Item Executors are nodes which are responsible for performing the actual code execution of work items created by Work Item Generators. Work Item Executors will monitor the blockchain for new blocks created by the Coordinator and extract the CIDs included within the blocks. They use the CIDs to recreate the files locally, using all of the relevant CID information in the Work Item to create identical versions of the code and file structures on their local machine.

Once downloaded locally, a Work Item Executor will run the same parser program that was used by the Work Item Generator and Coordinator Node. The parser validates that the code is safe to execute and will generate the required logs. If the parser validation passes, the node will send a message to the Coordinator to record the start time when execution started. Simply by running the program, a log file will be generated along with a solution file. The work items created by the Work Item Generators are designed to be deterministic and as such, when results are saved to IPFS, an identical CID will be created by each correct response. In addition, the unique log file will be generated and saved to IPFS. Both CIDs are submitted to the Coordinator to be evaluated along with other responses.

Work Item Executor nodes are perform execution actions while working within that role. It is possible for any Work Item Executor node to request to become a Coordinator Node as part of the process to select the next Coordinator Node. Once the work is complete, a new node is selected. All of the Work Item Executor nodes which participated are considered for selection and are chosen at random by the current Coordinator Node by producing a ranked list of the candidates. If a new work item is not selected within 5 minutes, the next

candidate on the list is able to select a work item. This process continues until a node that is within their appropriate window selects a new work item.

4.3 PROCESS FLOW

In the previous section we addressed the roles and capabilities of each of the respective nodes within the system. This section delves deeper into the process from start to finish. The Advanced Work framework uses a multi-stage approach which begins when a Work Item Generators create a program, written in python, that meets all security checks by a parser program. The parser program is used to analyze the code for vulnerabilities and ensure that it will be safe to execute by the remote nodes and looks for specific commands which could potentially be used in a way that can harm the remote machine.

Once an appropriate program and the associated documents are created, the files are uploaded to IPFS and the respective CIDs and other parameters are recorded in a JSON formatted configuration file. The configuration file is uploaded to IPFS for on final CID reference. The final CID is all that is needed to unpackage the job on another node for execution. The Work Item Generator monitors the blockchain in the Work Coordination Enclave to determine the current Coordinator Node and sends the CID to the Work Item Coordinator. If there are no pending work items, the Coordinator Node will select the work item as the target item for Work Item Executors will work on. However, if other items are already present, it will be added to a list of pending jobs. The selected work item is recorded as a transaction in a new block on the blockchain.

Worker Item Executors review the new block and extract the CID and uses it to download a local copy of the data from IPFS. The Work Item Executor uses an executor program to recreate the data and file structure from the CID references, performs a security check with the parser and prepares to begin execution of the code. Prior to starting the execution of the code, the Work Item Executor, sends a message to the Coordination Node to log the execution start time. Next, the Work Item Executor During moves into the execution phase where it runs the program and generates a local log. Once the execution completes the log and results file are uploaded to IPFS and the respective CIDs are recorded. The CIDs are submitted back to the Coordinator Node for validation and the Coordinator Node records a timestamp for when the data was received. Multiple Work Item Executors submit similar results to the Coordinator Node and when enough identical solution files have been received, the solution is accepted as correct. All of the Work Item Executors which provide correct solutions and logs within an acceptable time period are given a reward recorded in a new block along with the details of the new Coordinator Node selection list. Finally, the Coordinator Node sends the results back to the originating Work Item Generator. The process flow breakdown can be seen in Figure 4.2

4.4 FILE STRUCTURES

In order to facilitate a mechanism which is able to create code, store it on IPFS and recreate it for identical execution across multiple nodes, a standardized structure for file formats and folder hierarchies is needed. This section covers the required files and their purpose when used within the system. An overview of the files and where they are used can be found in Table 4.1. The files used by the system components are made up of python files,

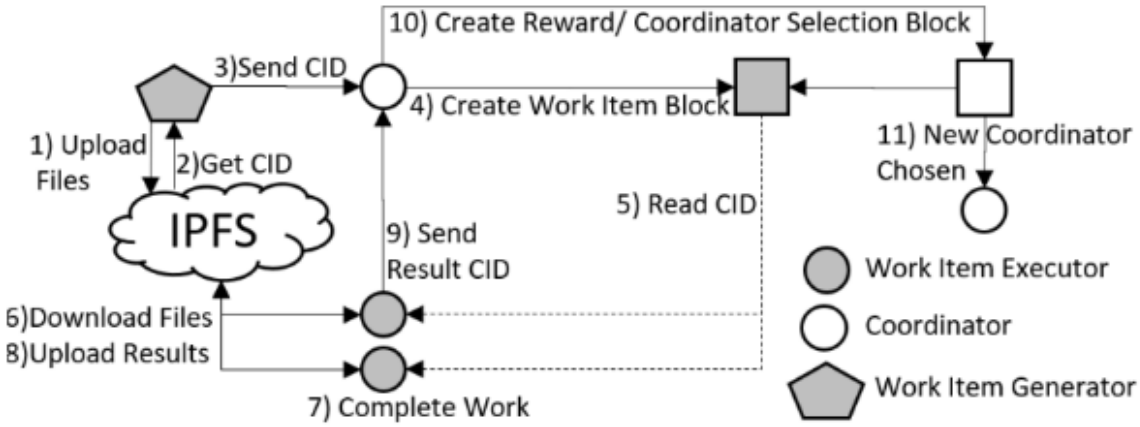

FIGURE 4.2: A sample work item workflow

JSON formatted config files, text files and any input files, which may have varying file types. Examples of acceptable file format types include .csv, .txt and .xls. The python files include execution.py, executor.py, parser.py, and startup.py.

Execution.py stores the code which will be executed remotely. Only one python file can be submitted so the file must contain all the required code and only reference authorized python packages. Executor.py is a file that is used to locally recreate the file structure and files required for execution and initiate the code execution. It acts as the engine for the system to execute the same code with the parameters and arguments provided in the configuration file.

Parser.py is used to check the code for compliance and ensure all required items are included. It also performs a security check by verifying that no unauthorized Python modules or commands are included. If a violation is found, an exception is thrown which prevents the execution of the locally created execution.py file by the remote system.

The final Python file is startup.py, which is used to ensure all required files are pulled

down from IPFS to the local system. It installs all required packages, runs the parser.py file and if no errors are encountered then runs the executor.py file, which runs the work item code stored in execution.py.

In addition to Python files, several files in JSON format are also used. The JSON files act a configuration files and store references to other IPFS CIDs used during setup and execution process. The reference.json file is used as the primary configuration file and store reference to IPFS CIDs for all the files that will be required for execution. The format for the reference.json file was briefly discussed earlier in Figure 4.1. Specifically, the file includes a reference to the CID for the requirements.txt file, the execution.py file, the executionconfigs.json, file the random seed value, the minimum and maximum time for execution, and a JSON array of input objects which need to be downloaded locally to execute the code.

The executionconfigs.json file contains a JSON array of execution objects which each include the parameters which are passed to the execution.py file for each iteration included. This allows the same code to be executed with different parameters while still being treated as a single work item. During execution, the executor.py file keeps track of logs generated during execution of the execution.py file. After the execution is complete, the logged results are saved into an output.txt file, the expected outputs are saved into the respective output files listed in the executionconfigs.json file. Each output is then saved to IPFS to retrieve a CID and the value for each file is stored in a finalresults.json file which is submitted to a Coordinator for verification.

TABLE 4.1: File names and descriptions.

File Name	Purpose	Use By: Work Item Generator (G) Coordinator Node (C) Work Item Executor (E)
references.json	Contains a reference to the IPFS CIDs for Executor.py, requirements.txt, and input files.	G,E
executionconfigs.json	Contains a list of one or more execution configurations which include parameters passed when the file is executed. Contains a list of the expected output types the files names.	G,E
finalresults.json	Contains a list of the output results and log files generated after successful completion of the code execution. Used by the Coordinator to verify result and by the Work Item Generator to retrieve verified results.	G,C,E
execution.py	Contains the code to be executed by the remote system.	G,E
requirements.txt	Contains a list of python libraries that need to be installed in order to execute the code.	G,E
output.txt	Used to compare results from multiple Work Item Executors to verify results.	C,E
restricted.txt	Used by the Parser.py file to check the execution.py file for compliance against restricted module or commands.	G,E
parser.py	Used to analyze the executor.py and executionconfigs.json files to verify they meet all requirements and standards.	G,E
startup.py	Used to create local copies of executor.py, executionconfigs.json, and all input files, and run the parser.py file to ensure the code is safe to execute locally.	E
executor.py	Used to read the executionconfigs.json file, execute the execution.py files, generate outputs and save results to IPFS.	E

4.5 EXPERIMENTATION AND RESULTS

The feasibility of the Advance Work Framework was tested in our previous work efforts [1]. In the research effort, a small-scale prototype consisting of 12 nodes hosted on AWS was used. Ten nodes acted as Work Item Executors, one acted as a Work Item Generator and one acted as a Coordinator. Each node was configured so they were not on a shared subnet to simulate distinct nodes distributed across the network environment.

Two scenarios were used to demonstrate the diversity of work items types. The first was a toy example based on the Game of Life simulation [36]. The second scenario was based on a modified version of a program that calculated survival predictions for a Titanic dataset [1]. The first acted as a proof of concept to demonstrate that code and be constructed in a way that supports security checks, logging and the ability for the code to be reconstructed and executed on remote machines to produce the same results. The program used the game of life scenario with 100,000 generations on a 1,000 by 1,000 grid. 10 nodes were created using Amazon Web Services(AWS) Elastic Cloud Compute (EC2) [2] instances with IPFS and the required Python files for parsing and execution. The proof of concept demonstrated that each node was able to take a starting CID reference and recreate the program locally, scan the files, execute the code and generate a log and and solution file before submitting the results to an AWS node designed to collect the results. After the functionality of the system was validated, the second scenario was created to test the diversity of the type of programs that could be submitted.

[1] https://www.kaggle.com/code/nadintamer/titanic-survival-predictions-beginner
[2] https://aws.amazon.com/pm/ec2/

The second scenario used a work item that was based on an existing implementation of the Titanic Survival prediction challenge on Kaggel.com. In this case, the goal was not to create the most accurate solution, but rather a solution which would complete the task and produce a result regardless of accuracy. The concept was based on an idea that users may not submit an optimal code solution as a work item, but rather any work item that they would not want to run locally using their own resources. The code was modified so that all references to commands or modules flagged by the parser were replaced with alternatives. In addition, the required annotations were added to each method for logging purposes. Once the code was updated and passed the validation performed by the parser the unmodified and modified versions of the code were both executed locally to validate the changes did not impact the results generated. After confirmation that the same results are generated, the Work Item Generator node was recreated using AWS and generated the work item to start the process.

A node operating as a Coordinator Node received the work item and created a simulated block to advertise the work item. The ten Work Item Executors successfully recognized the work item was available and began the process to download and execute the program. The ten nodes each had distinct levels of resources set during creation to test the ability to execute the code for various resources available from nodes. Work Item Executors which completed work, saved results to IPFS and sent messages to the Coordinator. To demonstrate the prototype, a maximum execution time was set for 2 hours and 30 minutes. All nodes which did not provide a solution within that time were ignored. In addition, the Work Item Generator required 5 correct responses to be considered valid per the configuration file generated. All nodes which provided correct responses at the end of the time were given a reward recorded

in a new block.

The prototype created as part of this research effort tested the three primary components of the framework. Of the ten nodes created as Work Item Executors, eight successfully executed the code and produced identical results. The two nodes which did not complete within the time frame suffered from faults due to insufficient memory to execute the large dataset used.

The Work Item Generator was configured to required 5 correct responses as a minimum value to prove the solution was correct. The maximum execution time was set for 2 hours and 30 minutes (9,000 seconds) was used in the configuration file. All nodes which generated identical results in the time period were treated as correct and rewarded. The eight nodes which completed within the timeframe were properly rewarded, through a balance update in a block created by the coordinator. The two nodes which did not finish were not rewarded as expected. Table 4.2 highlights the configurations for the ten Work Item Executors and their respective time used for setup and execution.

Finally, a random generator was used by an external program to select five nodes which had successfully completed work to simulate the process of committee selection. The node examined the blockchain for reward balances and selected those whose balance which was greater than 1 as the criteria for consideration. As expected, the external program only selected five nodes from the eight eligible nodes while ignoring the two which did not complete the work items.

TABLE 4.2: Execution using 10 nodes for real-world example.

	RAM / Storage / CPUs	Setup Time	Execution Time
Node 1	8 GB / 30 GB / 2	20 min	1 hr 30 min
Node 2	8 GB / 30 GB / 2	19 min	1 hr 30 min
Node 3	8 GB / 30 GB / 2	10 min	1 hr 30 min
Node 4	8 GB / 30 GB / 2	10 min	1 hr 30 min
Node 5	4 GB / 30 GB / 2	10 min	1 hr 50 min
Node 6[1]	4 GB / 30 GB / 2	8 min	1 hr 50 min
Node 7[1]	4 GB / 30 GB / 2	8 min	1 hr 50 min
Node 8[1]	4 GB / 30 GB / 2	8 min	1 hr 50 min
Node 9[2,3]	2 GB / 30 GB / 1	10 min	N/A
Node 10[2,3]	1 GB / 30 GB / 1	10 min	N/A
Notes: 1 Faster setup time possibly due to more nodes hosting files 2 Below IPFS minimum requirements 3. Insufficient memory to execute code			

CHAPTER 5

BLOCKCHAIN DESIGN

While the FAWAC architecture includes a novel way to support a free distributed computing solution through the advanced work framework, the capability is included as a method to create an entry criteria for consideration in the committee-based consensus mechanism. The FAWAC architecture is designed to create a secure and fair blockchain system as one of the core objectives of the network. In this chapter, discussion is shifted towards the concepts supporting the blockchain portion of the network within the Transaction Enclave.

The Work Coordination Enclave includes a blockchain-like structure which is used to record the order of work items selected, Coordinator Node selection and reward balances, however, it does not support the peer-to-peer exchange of value associated with traditional blockchain networks. The functionality of peer-to-peer value transfer occurs within the Transaction Enclave of the FAWAC architecture. This chapter focuses on the specific aspects of the blockchain within the Transaction Enclave. As the blockchain uses committee-based consensus, references to the committee selection process will be made, however, the specifics will be discussed in more detail in the next Chapter. For example, the structure of block data to record reputation scores is discussed here, however, how the data is used will be covered in the next chapter.

5.1 ACCOUNTS

In blockchain networks, access to the currency owned by a particular user is handled through public and private key pairs and associated digital signatures to verify the transfer between the users. There are two primary ways in which this is handled in modern systems. The first is through Unsigned Transaction Outputs(UTXOs) used by networks like Bitcoin [59] and the second is through an Account model used by Ethereum [16]. Each UTXO is tied to a unique Bitcoin address and is the resulting output of a transaction processed by the blockchain. Each UTXO is used as an input for another transaction and is completely consumed in the process. Each unique address requires a unique private and public key pair so an account owned by an end user often has many unique private keys to manage the UTXOs that belong to them, usually handled through a Hierarchical Deterministic(HD) wallet [5].

While the UTXO model was the original model used for blockchain networks, most networks now use the Account model. The Account model uses a single pair of private and public keys tied to a user address. The address for the user is used as a key value which references the balance for the user in some type of account state database system [6]. The model is popular because a user can be associated with a specified address that does not change. Most blockchain networks utilize Elliptic Curve Cryptography [43] for key generation and digital signatures over other methods due to smaller key sizes compared to other alternatives [71]. The FAWAC architecture blockchain uses a similar Account model, however uses mechanism used for key generation and transaction signing is different. One of the design considerations made while developing the FAWAC architecture was ensuring that the network would remain secure in a post-quantum computing age. The blockchain

portion of the FAWAC architecture uses digital signature mechanisms that are considered to be quantum secure, specifically an implementation of the Dilithium algorithm [32].

Similar to the Ethereum implementation of Accounts, an account can be owned by either an end user or a Smart Contract [16] Depending on the type of account, the specific data stored withing the account will change, however the format will remain the same. Accounts exist as JSON formatted data and are stored on IPFS. Additionally, the public key generated during the account creation process is also stored on IPFS, meaning that it can be easily retrieved and used to validate signatures using the CID generated creating an efficient key exchange process. The IPFS CID generated from the storage of the data is the account's resulting address.

The fields included in an account object include "content", "type", "owner" and "index". If the account belongs to a user, the content field contains the CID for the public key for the user, otherwise, it includes a CID for the contract code that has been uploaded to IPFS. The type field represents whether an account belongs to a human user, represented by 0, or a smart contract, represented by 1. The owner field is primarily used for smart contract deployment and is used to indicate the CID address for the account that deployed it. If the account is a user account, the field includes a duplicate value for the content. The index field is used to record how many versions have been created by the user and is used to create a unique value when otherwise similar data is submitted during the account creation process. It is possible to create a smart contract with identical code that is deployed multiple times. However, a user may want to have each instance to have a unique address and a unique state associated with it. If the account data only included the content, type and owner values, any time a user deployed the same code for a smart contract it would result in an identical CID

which could cause unintended consequences. The addition of the index field ensures that the account data is unique and collisions for account addresses do not occur. An example account structure can be seen in Listing 5.1.

```
{
  "content": "Qmw8bg4...UJpm",
  "type": 0,
  "owner":"Qmw8bg4...UJpm"
  "index":0,
}
```

LISTING 5.1: Example Account Structure (CID Truncated)

5.2 SYSTEM DESIGN

Each node within the FAWAC architecture blockchain operates an IPFS node to allow interconnection to the other enclaves and support the exchange of data between nodes. The code for all nodes is written in Python with multiple files supporting the various responsibility of the nodes. An IPFS helper file is created to handle integration with the node's local IPFS instance to include uploading files, calculating CID values, and retrieving the raw data stored on IPFS for use by the other modules. Each node also operates a client/server module which actively listens for new connections and passes the data on to known peers. The client/server module is used by a miner program to initiate connections to peers when the node is selected as a block producer and creates a new block. The client/server module also tracks pending transactions to be evaluated by the miner and stored until processed by an the miner or another node on the network. Additionally, python programs are included to

define the structure of the block transaction types and the logic used for their validation. Finally, a smart contract runner program is also included to handle the extraction of smart contract code and the execution of specified methods indicated in transactions.

Each node on the network contains a wallet program which manages the account balance for the node. When new blocks are created, a reward is given out to nodes that participate in the committee. To access the rewards, the node uses their associated wallet program to create, sign and submit transactions. While a node participating in the mining process needs a wallet, not every wallet needs to belong to a node participating in mining. Wallets can be created for any user to send and receive currency or interact with smart contracts.

The blockchain maintains a record of changing states within the network. The states are managed by a JSON formatted record for all accounts that have interacted with the blockchain. The structure of the account state uses a nested JSON structure where the keys are associated with the account address and the value is another JSON object containing a "Value", "Reputation", "Data" and "Nonce" field. Value is a representation of the amount of currency the account owns. "Reputation" is used for user accounts that participate in the consensus process and is used to record the score used for committee selection calculations. The field is unused for smart contract accounts. The Data field is not used for human accounts but includes a CID that holds the data state values for a smart contract. Finally, a "Nonce" is used to record the last transaction made for the account. It is used by human users to determine if a transaction should be processed. Each transaction should only be processed once. Each transaction includes a nonce value when it is created. To prevent double processing of a transaction, transactions are only processed if the nonce value is higher then the nonce value recorded in the account state for the respective account. For

smart contracts, the Nonce value is used to record the number of times the contract has been interacted with. An example account state with two human owned accounts and one smart contract can be seen in Listing 5.2

```
{
  "Qmw8bg4...UJpm":
  {
    "Value":10,
    "Reputation":150,
    "Data": "",
    "Nonce":3
  },
  "QmwZhrt...od5W":
  {
    "Value":7,
    "Reputation":90,
    "Data": "",
    "Nonce":2
  },
  "QmwfiYs...88Ds":
  {
    "Value":0,
    "Reputation":0,
    "Data": "QmnnLa...xxC2",
    "Nonce":10
  },
}
```

LISTING 5.2: Example Account State (CID Truncated)

The integration of the FAWAC network with the IPFS network enables a simplified exchange of data between nodes. As long as the IPFS CID is known, and at least one node maintains a copy of the data, the data is eventually retrievable. However, in a blockchain network that is attempting to validate transaction data as quickly as possible, eventually retrievable is not an acceptable standard. Each node in IPFS has a unique peer address that is created when it is first initialized [14]. While connected peers can be related to previous interactions, IPFS also has a mechanism in place to force a node to be a peer. When peer connections exist, file retrieval is fast, especially if the file is hosted by several peer nodes. The FAWAC architecture takes advantage of this mechanism and integrates it into the design in several places. First, the peer connection details; to include IP address, port and IPFS ID, is recorded for each node in the current committee. Each member of the committee establishes a peer connection to each other member once selected. If a user wants to submit a transaction, the wallet program selects one of the nodes on the committee and establishes a peer connection prior to submitting the transaction to ensure it can be retrieved. Committees are transient in nature and change after every 10 blocks. Maintaining the connections to all of the committees that change so regularly could be a burden on end users, so the FAWAC architecture also includes another type of node service called a Gateway Node. Gateway nodes are stand alone nodes that do not participate in validation, however they establish a connection to every committee whenever it changes. If an end user connects to a Gateway Node, the data can be relayed to the committee on the user's behalf without establishing a connection to the network directly.

Integration with IPFS also creates an additional benefit by allowing nodes to selectively prune data to limit storage requirements. This allows for three different node types that can

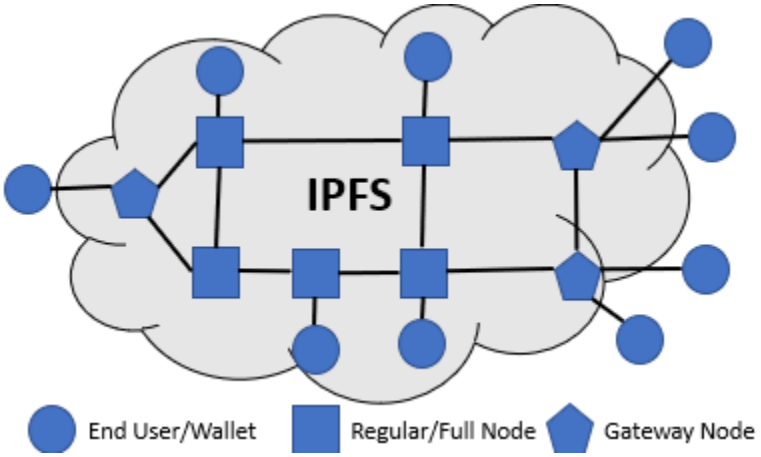

FIGURE 5.1: Node type and system design

be operated. Figure 5.1 highlights the node types and high level system design. The first Node type has already been discussed as the Gateway node. The next node type is the Full Node which maintains a record of all transactions, account states, account addresses and blocks ever created by the network. Nodes operating as a Full Node can also advertise their IPFS address to other nodes in order to streamline peer connections, increasing the speed at which data can be retrieved. This is the most resource intensive version of the available nodes. Full Nodes are able participate in the network's consensus mechanism and earn rewards from participation.

The final node type is the Regular Node. The primary difference between a Regular Node and a Full Node is that a Regular node is only required to maintain IPFS data processed while it was selected to participate as part of a committee as well as the first and last blocks in an epoch. An epoch is defined as the series of blocks which are validated by the same committee and lasts for 10 blocks. A new committee is selected during the creation of the last blocks in the epoch and includes a reference to the next committee selected. The Regular Nodes that are not part of the current committee are not required to perform validation on

the first or last block in the epoch, but are required to download copies of the contents to act as an additional storage point for the files on IPFS. Regular Nodes can also participate in the network's consensus mechanism and store all data directly related to their participation in their local IPFS instance when selected to be part of a committee.

The FAWAC architecture supports smart contract execution in addition to regular peer-to-peer transactions. Since the structure of FAWAC blockchain is very different from other implementations, the system is not compatible with the most commonly used smart contract framework, the Ethereum Virtual Machine(EVM) [16]. A new smart contract framework was designed to incorporate smart contract capabilities into the FAWAC architecture. All contract code is written using Python and the code is executed through a smart contract runner program that is included as part of the core code ran by nodes. The smart contract runner has several core responsibilities to include reading data from IPFS, executing code and updating the state of a contract following execution. The typical process flow for the smart contract runner involves the following steps.

1. Read the account data from IPFS using the smart contract's account referenced in the transaction.

2. Use the Content value of the Account Data to read the contract Code from IPFS and save to a temporary file locally on the node.

3. Execute a parser program to verify the contract meets all requirements.

4. Use the smart contract address to extract the smart contract state from the account state.

5. Use the __import__() python method to dynamically import the temporary file created.

6. Create an instance of the smart contract class.

7. Use the smart contract class instance to call the built in method to load the account state, passing in the data extracted in step 5.

8. Extract the DATA value from the transaction and parse it into a Python dictionary and determine what method and parameters to use.

9. use the instance of the smart contract class and the Python getattr() function to call the requested function with the provided parameters.

10. Save any return two arrays containing accounts and balances to update for the blockchain account state. If the contract call does not send currency to other accounts, two empty arrays are returned.

11. Use the instance of the smart contract class to read the updated contract state

12. Store the new contract state on IPFS and use the CID to update the reference in the account state.

The operations executed by the smart contract runner are wrapped in a Python Try/Except block and if at any time the event fails, the transaction is reverted without changes to the contract state or account state.

While Python is not a compiled language and the structure we implements supports dynamic code execution, some specific requires must be met in order to ensure the safety

of nodes executing code and to minimize the chance that execution will fail. The parser program is similar to the one used to support the advanced work framework. The parser reads the code and checks for the inclusion of modules that may pose a risk to the network. The parser also makes sure that the minimum required methods are included. At a minimum the getState(), loadState(), __init__(), and initialize() functions must be included with the default implementation as part of a contract class. A basic example of a minimal contract can be seen in Listing 5.3. In the example, an additional custom function is added to the basic required functions. With the exception of the initialize function the structure should be the same for all contracts. the initialize function can vary from user to user based on what values are created initially when the contract is deployed.

```
class mycontract:
    def __init__(self):
        self.state = {}
    def initialize(self):
        self.state["sample"] = "hello world"
        self.state["counter"] = 0
    def loadState(self,state):
        self.state = state
    def getState(self):
        return self.state

    ##### CUSTOM FUNCTION BELOW THIS POINT

    def updateCounter(self,value):
        self.state["counter"] = value
        return [],[], None
```

LISTING 5.3: Basic Smart Contract Code

The load_state function takes the a JSON object as a parameter which represents the current state of the contract and uses it to create a Python dictionary representation of the account state. The current state of the contract is retrieved from the previous block and this value is passed as the argument for the loan_state() function. It is called by the block producer node prior to making any calls to the contract that are defined in the transaction. Similarly, the get_state() function is called after the function identified in the transaction completes to record the updated state value to be uploaded to IPFS and stored in the newest account state.

When a transaction interacts with a smart contract, some additional data is included as part of the transaction. The data is a JSON format file that includes a reference to all of the information needed by the smart contract to execute a specified function. The information includes a reference to the function being called and an array of parameters in JSON format using keys called "Name" and "Value". For example, a smart contract function called "Set_Message(m)" that takes an argument "m" to update the state of a message variable within the contract's account state to "hello world" could be described in Listing 5.4. The resulting data that would be included in a transaction can be seen in Listing 5.5. When calling smart contracts, the parameters include both name and value because keyword arguments are used when calling the functions to ensure proper assignment of parameter values.

```
def Set_Message(m):
    state["message"] = m
```

LISTING 5.4: Sample Smart Contract Function

```
{
  "function" : "Set_Message",
  "parameters" :
```

```
    [
      {
        "name" : "m",
        "value : "hello world"
      }
    ]
}
```

LISTING 5.5: Sample Data Submitted In Transaction

All of the other components of the network do not have much value if the network is not able to achieve consensus on the validity of transactions submitted. This makes the consensus process one of the most vital components of the FAWAC architecture. The FAWAC architecture blockchain uses committee-based consensus mechanism that considers the reputation of nodes as part of the selection process. To support a reputation system, a mechanism is used to record behavior and reward and penalize nodes properly. However, traditional block chain networks to not contain a mechanism to record malicious behavior. If an invalid block is proposed, other nodes ignore the block and move on to the next block they receive. Ignoring the invalid data keeps the network stable by preventing the invalid data from being included, however, it does not do anything to prevent the node that sent the data from attempting to send it again. Even in committee based system where voting occurs, if enough nodes vote against a block to reject it, the block is disregarded and the next block is processed. Only blocks that are approved by the committee are ever acknowledged. To be able to record voting behavior of nodes, the structure of the blocks and how they are presented needs to change. Specifically, the FAWAC architecture uses a multi-block structure to enable historical tracking and validation of committee-based consensus

records. The blockchain splits a block into three independent, yet inter-related parts called the Data Block, Validation Block and Meta Block. Each is interconnected with references to each other and collectively what is considered each round when a new block is created.

5.3 BLOCK TYPES

In traditional blockchain networks, only a single node is produced. While this approach works will in systems where a single node is chosen to create the next block, it creates issues when trying to use a committee-based consensus where honesty of nodes is important. Since only one block is created, once the block has been created and sent to other committee members, there is no way to update the block data to store the votes without changing the hash of the file in a way that prevents manipulation. To resolve the issue, we incorporated a block structure that consists of multiple sub-blocks created each round.

In order to support a reputation weighted committee-based consensus mechanism, a scoring system needs to be integrated. Whether a committee member is treated as honest and malicious is based on a simple majority vote for a pending block based on validation criteria. Initial attempts used Four sub-blocks, which included a data block, a null block, a validation block and a meta data block. Later efforts combined the data and null blocks as the null block was a special case wherein no transactions were processed by the chosen validation node within a given time frame. The Data sub-block contained transaction data. The second is a Validation sub-block, which record the votes from committee member. Finally a Meta block is used to record the account and reputation state changes. In each round, either a Data block or Null block will be created while a Validation and Meta block

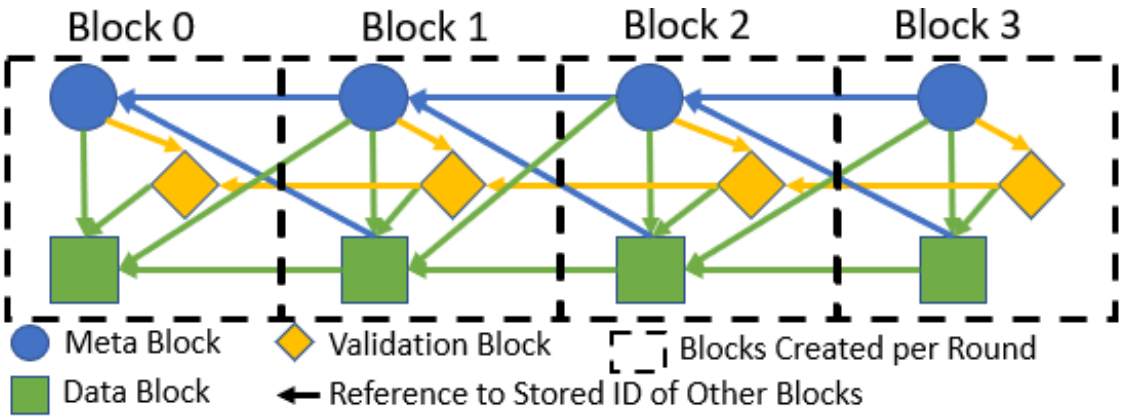

FIGURE 5.2: Block relationships

is created each round regardless. The blocks point to each other in overlapping ways for additional validation purposes. Figure 5.2 shows the relationship between the blocks.

The Validation block points to the Data sub-block and records the signed votes from each committee member. By recording the signed votes, it provides an irrefutable and verifiable record of which nodes agreed to or rejected the proposed Data block. Based on if the majority of nodes voted to accept or reject, a reputation score is adjusted, with those in the majority receiving a positive boost and those voting in the majority receiving a penalty.

The Meta block was the final block type we created and acts as a type of external header for the other block types. Each Meta sub-block points to the previous Meta sub-block but also points to the Data sub-block and the Validation sub-block. Based on the results of the Validation sub-block, the Meta sub-block will update the reputation score for each committee node as well as store the updated balances for all accounts based on transactions stored in the Data Block. If a Null sub-block is created for the data sub-block, a Validation sub-block is still created to validate the conditions for the creation of the Null sub-block. The

Meta sub-block will not update account balances but will update the penalty for the original block creator and nodes that voted to reject the Null sub-block.

Block data is saved using JSON format and stored on IPFS by nodes. Each blocks included references to the CIDs for the stored data. Nodes in a validation committee are responsible for storing a local copy of the files referenced by the CIDs, thus ensuring the data persists. These nodes have an incentive to store the files, since the blocks contain references to their respective rewards for participating in validation. Sample block JSON structures (with CIDs truncated to save space) can be seen in Listing 5.6 Listing 5.7 and Listing 5.8

```
{"Data":
  {"Timestamp": 161007200.0,
   "Height":10,
   "BlockRef":"QmZtspk...o7st",
   "ValidationRef":"QmfPosd...n4Dt",
   "EpochID":2,
   "PreviousEpochCID":"QmFtk7t...E52w",
   "PreviousMeta":"QmEnRBj...M4Qsk",
   "PreviousBlock":"Qmw8bg4...UJpm",
   "AccountState":"QmE6vry...3dpV"
   "ReputationState":"QmyykBg...bBcq"
   "PendingMembers":"Qm97fSm...UCfk",
   "CommitteeID":"QmCh18n...pPst",
   "NextCommitteeID":"QmCh18n...pPst"
   }
 "Signature": "39e0ef881b...12fa75"
}
```

LISTING 5.6: Example Meta Block (CID and signature truncated)

```
{"Data":
  {"Timestamp": 1601000200.0,
   "Height":10,
   "BlockRef":"QmZtspk...o7st",
   "MinerID":"Qmwf7x2...LnN2",
   "Status":"Accepted",
   "Votes":
    ["Qm8rsd...8cqb","QmM6Ma...bp7a",...]
   }
 "Signature": "ba6e43f611...58ae2f"
}
```

LISTING 5.7: Example Validation Block (CID and signature truncated)

```
{"Header":
  {"Timestamp": 160995200.0,
   "Height":10,
   "PreviousBlock":"Qmw8bg4...UJpm",
   "ValidationRef":"QmfPosd...n4Dt",
   "EpochID":2,
   "PreviousAccountState":"Qmgb7wW...2hyV"
   "AccountState":"QmE6vry...3dpV",
   "TransactionsCID":"QmgTrxv...dpvY",
   "PreviousEpochCID":"QmFtk7t...E52w",
   "PreviousMeta":"QmEnRBj...M4Qsk",
   "MinerID":"Qmwf7x2...LnN2"
   }
```

```
  "Body":
   {"Transactions":
    ["QmwxxY..17Hdefp", "Qms7jN...9uMx"...],
    "Signature": "2ef19d...5h6a8aa"
   }
}
```

LISTING 5.8: Example Data Block (CID and signature truncated)

5.4 TRANSACTIONS

Transactions are the primary way in which users exchange currency values between each other or interact with smart contracts. Transactions are created using a standardized format and signed using the private key of the user and are processed sequentially based on whichever ones are selected by the current block producer. Transactions are used to update account states, but only if the block has been accepted as valid. Three types of transactions are supported by the FAWAC architecture blockchain, each which shares the same fields, but are processed differently based on types. Regardless of the type, the structure of a base transaction is the same and is made up of 7 fields. Transactions are stored as JSON format files and include a "To" field, a "From" field, an "Amount" field, a "Nonce" field, a "Type" field, "Data" Field and a "Signature" field. The three types include a peer-to-peer currency transaction, smart contract transactions and committee selection request transactions.

The transaction To field is used to identify the account address of recipient, whether it is another user or a smart contract. If the type of transaction is a committee selection request, the to field includes the account address of the sender. The From field holds the

account address of the sender and is the same for all transaction types. The Amount field includes the total value being transferred between the two accounts. The Nonce acts as a counter that is incremented each time the user creates a transaction. If a Nonce value is below what is included in the account state, the transaction is considered invalid. The Type field is used to identify the type of transaction where 0 is used for peer-to-peer, 1 is used for smart contracts and 2 is used for committee requests. The Data type can be left empty for peer-to-peer transactions between user accounts, but could include a message if the user wants since the field is ignored for peer-to-peer transactions. The Data field is used for smart contract interactions, by providing JSON formatted data referencing the method being called and the required parameter values. If the transaction is a committee selection request, the Data field includes an IPFS CID for the peer-to-peer connection details for the node. The connection details are JSON formatted data that contain the account address, IP address and port for the node and the IPFS connection ID. A example connection detail file can be seen in Listing 5.9.A sample peer-to-peer transaction can be seen in Listing 5.10, a sample smart contract transaction can be seen in Listing 5.11 and a sample committee selection request transaction can bee seen in Listing 5.12.

```
{
  "address": "Qmw8bg4...UJpm",
  "server": "3.13.252.251:5000",
  "ipfs":"12D3Koo...jwwz1oeC2
}
```

LISTING 5.9: Sample Connection Data For A Node

```
{
  "To" : "QmwZhrt...od5W",
```

```
  "From": "Qmw8bg4...UJpm",
  "Amount": 10,
  "Nonce": 1,
  "Type": 0,
  "Data":"",
  "Signature": "ev9evac....3fbba"
}
```

LISTING 5.10: Sample Peer-To-Peer Transaction

```
{
  "To" : "QmwfiYs...88Ds",
  "From": "Qmw8bg4...UJpm",
  "Amount":0,
  "Nonce": 2,
  "Type": 1,
  "Data":
  {
    "function" : "Set_Message",
     "parameters" :
      [
        {
          "name" : "m",
          "value : "hello world"
        }
      ]
  },
  "Signature": "3fcsnn....3fafc4"
}
```

LISTING 5.11: Sample Smart Contract Transaction

```
{
  "To" : "Qmw8bg4...UJpm",
  "From": "Qmw8bg4...UJpm",
  "Amount":0,
  "Nonce": 3,
  "Type": 2,
  "Data": "Qmjnsm3...aNur
  "Signature": "08ec7ce....1fe13f"
}
```

LISTING 5.12: Sample Committee Request Transaction

CHAPTER 6

COMMITTEE SELECTION & SECURITY

The advanced work mechanism for FAWAC creates a system that reduces the possibility that malicious actors can successfully execute a Sybil attack against the network, by imposing a cost before a node can be considered for committee selection. The process is only part of the security mechanisms used for committee selection. If only the advanced work mechanism is in place, it is possible that over time a collection of malicious nodes could have a large enough presence to be selected for a committee and disrupt the network.

The FAWAC architecture takes this into consideration and additional layers to the committee selection process within the Transaction Enclave to further minimize the security threat. First, FAWAC uses a reputation-based selection mechanism that is based on a node's previous behavior. By capturing voting behavior for nodes, a reputation score is maintained that increases when a node votes in the majority and decreases otherwise. The reputation is used as an input for a weighted calculation that also incorporates verifiable randomness. Additionally, only a small subset of nodes selected for the committee are chosen to propose a block. Of a committee size of 101 nodes, only 10 will be selected to produce blocks. The remaining nodes in the committee are responsible for evaluating proposed blocks and voting to accept or reject.

The chances of a malicious node being selected as a block producer are further reduced, by first ranking committee members by reputation, and choosing the 10 nodes with the highest reputation score for block production. Finally, the FAWAC architecture introduces a concept called "Adversarial Consensus" which identifies and penalizes nodes which vote to accept without performing any validation. In order for a malicious actor to disrupt the network, they must first produce enough work as a Work Item Executor to have 51 individual nodes eligible within the Transaction Enclave. Next, those nodes must be selected by as committee members enough times and act honestly so that the reputation score for at least one member is within the top 10 of selected nodes. If the number of malicious nodes in the committee is greater then the number of honest nodes in this condition, the network can be disrupted. However, due to the random nature of some aspects of the committee selection mechanism, many more than 51 nodes would need to be included to increase the odds of being selected to high enough levels. In our research and simulation experiments, we have shown that with the adversarial consensus mechanism in place, some scenarios existed where the network is secure when only 40% of the total nodes in the network are honest [2]. The specific details and results of the simulation experiments are discussed later in this chapter.

6.1 COMMITTEE SELECTION

The selection process starts at the end of each epoch, which is considered to be the creation of 10 blocks. Nodes wishing to participate in the next epoch send transactions which include the node's, IPFS ID, IP address, port it is listening on and their account address. The transactions are then stored on IPFS and the CID is submitted to the current committee

where the details are extracted from IPFS and it is verified. If valid, the CID is added to a list of pending members. In the last block of the Epoch, a committee selection process runs and records the new committee value as part of the Meta block data.

The selection criteria incorporates randomness at multiple levels in order to ensure the process was as free from bias as possible. To ensure the results can be verified a random seed based on the CID from the previous meta block is used with the random calculations. At the time the calculation occurs, the seed can not be predicted since the CID cannot be known until the previous block was created. However, since it consists of data that is already stored in the records of the blockchain, the calculations are able to be executed and verified again at a later time using the given seed.

The list of pending members is extracted from the IPFS and each member in the list is iterated over to extract their current reputation score stored in the latest Meta block. For any node without a score, a 0 is added to the list and all negative scores are removed. The system calculates an average from the remaining scores to use as a score modifier. A base score of 100 is used and the modifier is added to the base score to create a shared baseline score. A target value is then calculated as 75% of the shared baseline.

For each pending member, their respective reputation score is applied to the shared baseline. For honest members, this results in a positive increase while it results in a decrease for malicious members. A loop is initiated until all members of the committee have been selected. Within the loop, a random index is selected and the node at that index is evaluated. Evaluation consists of a random number between 0 and the node's adjusted score being generated. If the random number is higher then the target score, the member is chosen, otherwise the process continues within the loop and a new index is randomly selected. Once

all committee members are selected, the list is sorted using the reputation score such that the highest scores are at the beginning of the list. The final list of new members is stored on IPFS and a reference is added to the Meta block. Pseudo code for the process can be seen in Listing 6.1

```
positive_scores, selected_member = []
For Member in Pending_list:
  if Member.score ==0:
    positive_scores.add(0)
  if Member.score >0:
    positive_scores.add(Member.score)
  baseline = avg(positive_scores) +100
  target = baseline * 3 / 4
While selected_members < committee_size:
   random_index = new Random(0,size_of_pending)
   member_score = baseline + Pending_list[random_index].score
   random_number = new Random(0,member_score)
   if random_number > target:
     selected_members.add(Pending_list[random_index]
return selected_members
```

LISTING 6.1: Committee Selection Process Pseudo Code

6.2 LAZY AGREEMENT PROBLEM

The committee selection algorithm is used to minimize the chances that a malicious node is selected to be part of the committee. However, due to the random nature of the algorithm it is still possible a malicious node can be selected. The algorithm reduces the presence of

malicious nodes as was demonstrated in a simulation conducted during research efforts [3]. In the simulation, two scenarios were explored using. One instance executed an scenario where all nodes were honest and acted as a baseline while, the other scenario included malicious nodes. During execution, reputation scores were properly updated for committee members based on if they were set to act maliciously or honestly. The selection process included multiple random calculations and not all nodes were selected at the same rate. Every time a node was selected, a record corresponding to the node's respective ID was incremented. Afterwards, the data was compiled into a list which grouping nodes having the same number of times which they were selected. The count for each selection amounts was graphed using the range of selection rates on the X axis and the number of nodes in each respective selection rate on the Y axis.

The rates of selection for honest scenario can be seen in 6.1 and appear to mirror a traditional normal distribution for selection rates. Conversely, the scenario which included malicious nodes can be seen in 6.2 and includes a large spike in the number of nodes only selected less than 10 times. The spike correlates with the number of malicious nodes which had a higher probability of being rejected from a committee based on penalties applied. All nodes are considered honest initially and start with a reputation score of 100. Therefore, even malicious nodes were initially selected at a rate consistent with honest nodes until a large enough penalty was applied to impact selection rates. However, as the system runs for a longer duration, selection rates begin to diverge.

The reduction in the number of malicious nodes selected indicates that the likelihood of a committee made up of a majority of honest members is more likely than one that includes a majority of malicious one. While this is beneficial, it does create a potential for a new risk

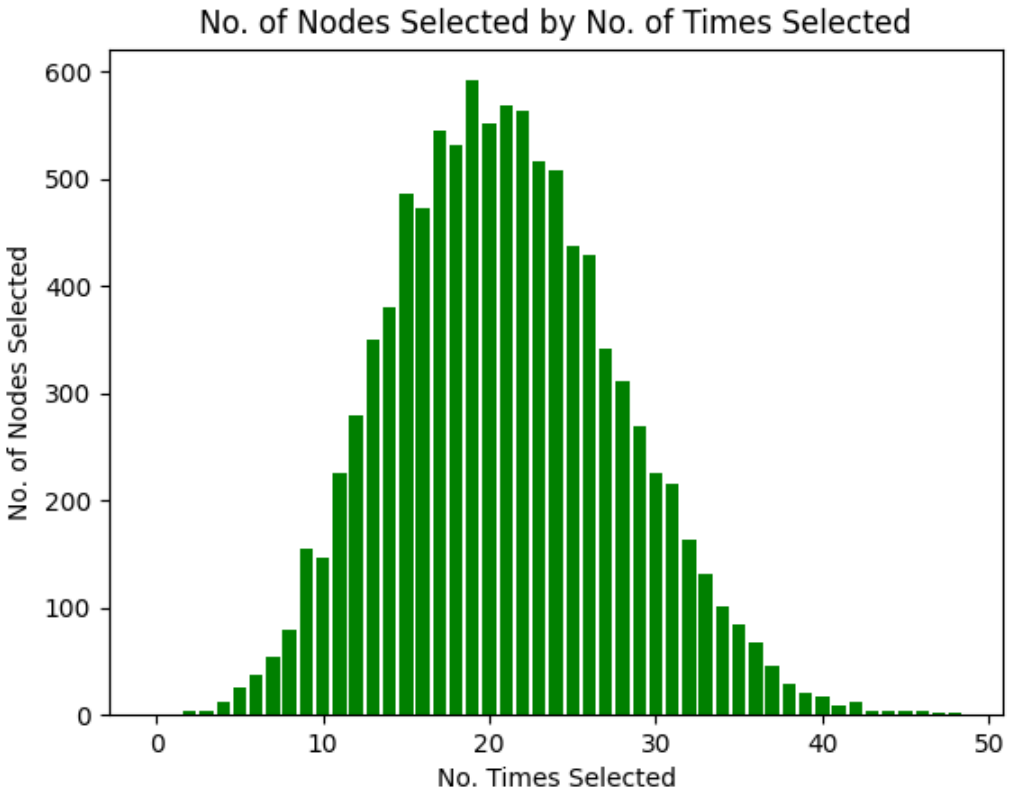

FIGURE 6.1: Count of nodes by common selection rate (honest scenario)

if nodes attempt advantage of the assumption that the committee is likely to be honest. It is possible for a node to be configured in such a way that the node does not perform any validation but, instead votes to accept by default with the assumption the block will be accepted more often than not. We defined this behavior as Lazy Agreement and identified it as a security risk for committee-based consensus mechanisms. The FAWAC architecture takes the Lazy Agreement problem into consideration with the Adversarial Consensus mechanism, however before it is defined, the full risk of the Lazy Agreement problem should be understood.

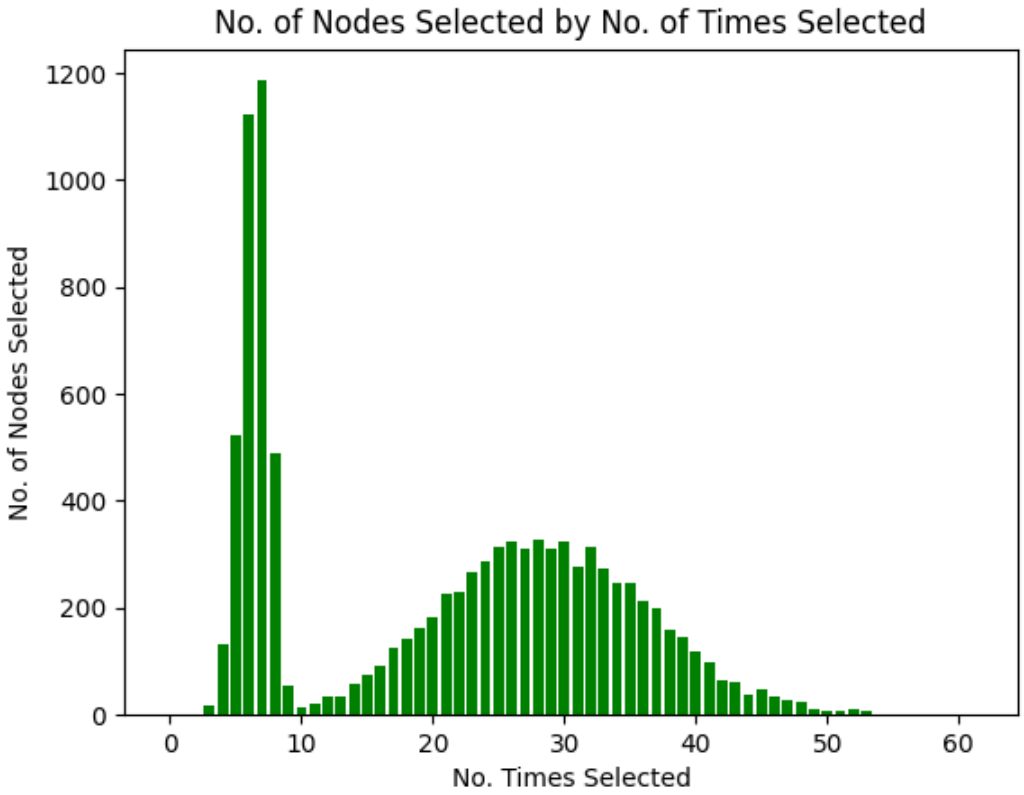

FIGURE 6.2: Count of nodes by common selection rate (malicious scenario)

Lazy nodes are distinct from malicious nodes in that malicious nodes actively attempt to disrupt the network while lazy nodes only attempt to save computational power by skipping validation based on an assumption that the network is secure and honest data is being proposed. Without the presence of malicious nodes, lazy nodes do not present a risk to the network. However, when malicious nodes are present, and are able to be selected as block producers, lazy nodes have an amplifying effect on the malicious node's weight.

To highlight the lazy agreement problem it is helpful to outlined a sample protocol for a blockchain that would potentially be vulnerable to the Lazy Agreement problem. If we

assume that $p <= c <= n$ where n represents the total number of nodes in the network, c represent the number of nodes chosen for the committee and p represents a subset of the selected committee who propose blocks in a round-robin manner. The steps of the protocol can be defined below:

1. A Committee of c nodes is chosen as a subset of n with p block proposers are selected from c. The remaining members of c are given a validation role.

2. The block proposer creates a candidate block and sends to all c nodes for validation while each node in c validates the block and sends back an "Accept" vote if the data is valid or a "Reject" vote if invalid.

3. The block proposer counts votes and if more "Accept" votes are received such than "Reject" votes, the block is created, otherwise, after a short time a recovery mechanism is initiated.

4. After p nodes propose blocks, a new committee c is selected from $p - c$ and the block is finalized.

In a network that allows lazy agreement, lazy nodes will vote to accept regardless of the block data proposed. Blocks proposed by an honest node have a better chance of being accepted, since the lazy nodes will always vote to accept a blocks. However, lazy nodes will have the same impact for malicious block proposers by creating additional vote power in support of the malicious block producer. For example, if a committee of 11 nodes is chosen with 5 being honest, 5 being lazy and 1 being malicious, it will be impossible for the malicious node to disrupt the network by rejecting honest blocks. However, if chosen

as a block producer, the combined vote power of 1 malicious node and 5 lazy nodes results in the malicious block being accepted by a simple majority. Lazy nodes amplify the vote weight of the block producer whether the block producer is honest or malicious.

6.3 ADVERSARIAL CONSENSUS

To address the Lazy Agreement problem, we incorporate the Adversarial Consensus mechanism into the block production process. Adversarial Consensus does not directly apply to the committee selection mechanism, but rather, provides opportunities to identify nodes performing Lazy Agreement and penalize the nodes similar to the way malicious nodes are penalized.

Adversarial Consensus incorporates a random flag that can be verified by nodes after a block is created and is included as part of the Meta block structure. When the flag is toggled the block producer is authorized to produce an invalid block and submit it to the committee. The block producer selects a random nonce value to include as part of a seed for pseudo random number generation of a value between 1 and 100. If the random number is above a specified threshold, the node has the option, but is not required, to submit invalid data for the block to committee members. Nodes are unable to determine at the time the block is proposed if the block proposer is authorized to produce invalid data. If the block proposer is authorized to act adversarial, nodes which reject the block and provide a correct "reason code" are given a positive update to their reputation scores. If the nodes fail to reject the block or provide an incorrect reason, they are penalized. For adversarial blocks, even if the majority of nodes vote to accept, the state of the blockchain is not changed in order to prevent

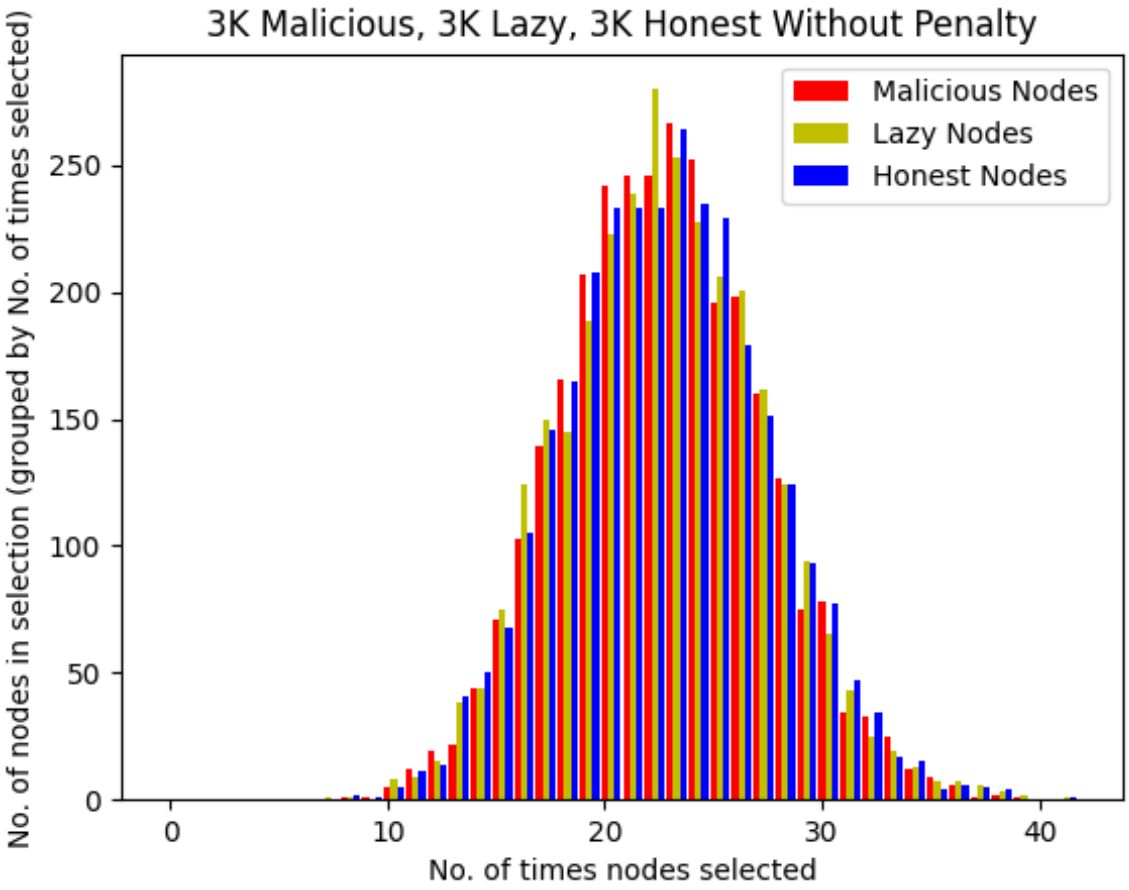

FIGURE 6.3: Node selection rates without penalty system.

the invalid data from being entered. After the block is finalized other nodes can verify that the adversarial conditions were met by recomputing the random number using the nonce values stored in the Meta block data. When the Adversarial Consensus mechanism is used, the selection rates of lazy nodes is also decreased along with malicious nodes. as seen by a shift to the left for lazy node selection numbers and a slight shift to the right for honest node selection numbers in Fig. 6.5

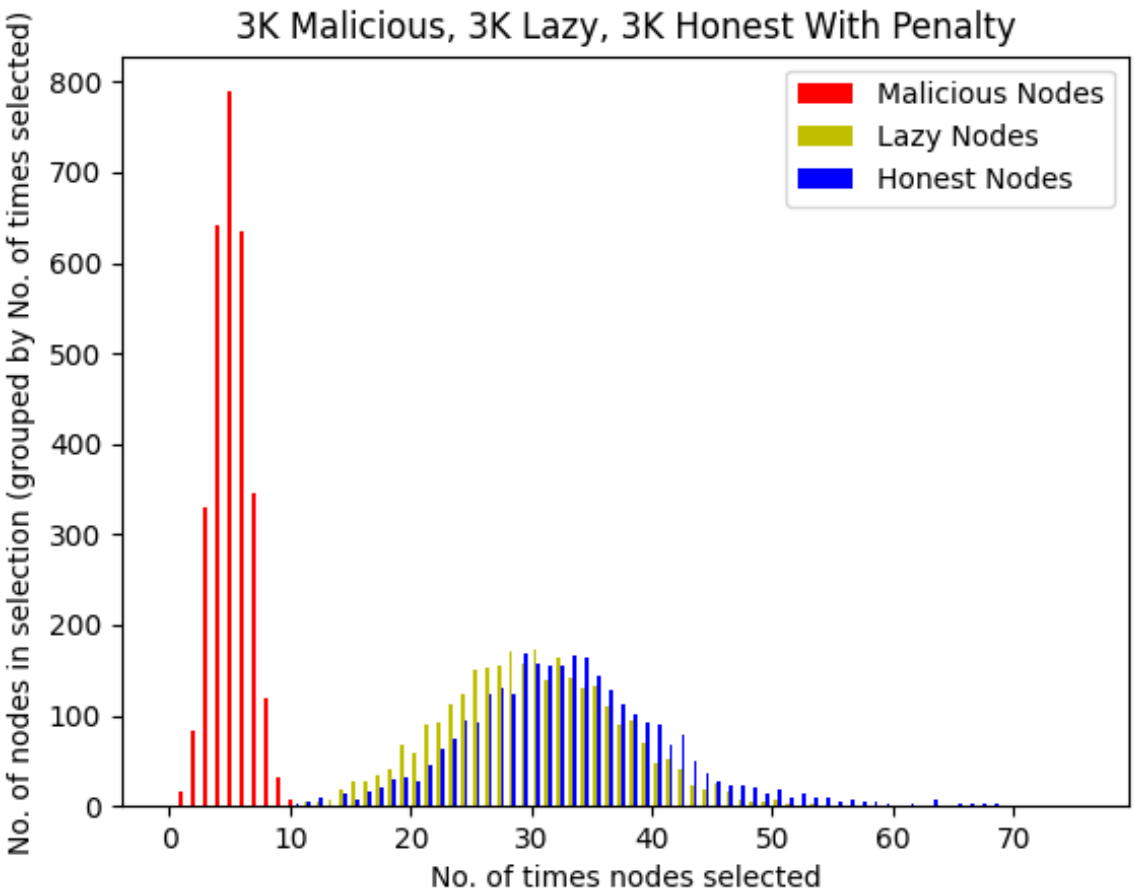

FIGURE 6.4: Node selection rates with penalty system.

6.4 EXPERIMENTATION AND RESULTS

To verify that the Adversarial Consensus mechanisms is effective at reducing the presence of lazy nodes as well as malicious nodes, several experiments were conducted. To highlight the impact of lazy nodes being present, the committee selection criteria was modified, removing the sorting at the conclusion of the process. The purpose of the modification was to increase the chances of a malicious node becoming a block producer, which is a requirement for the Lazy Agreement problem to occur.

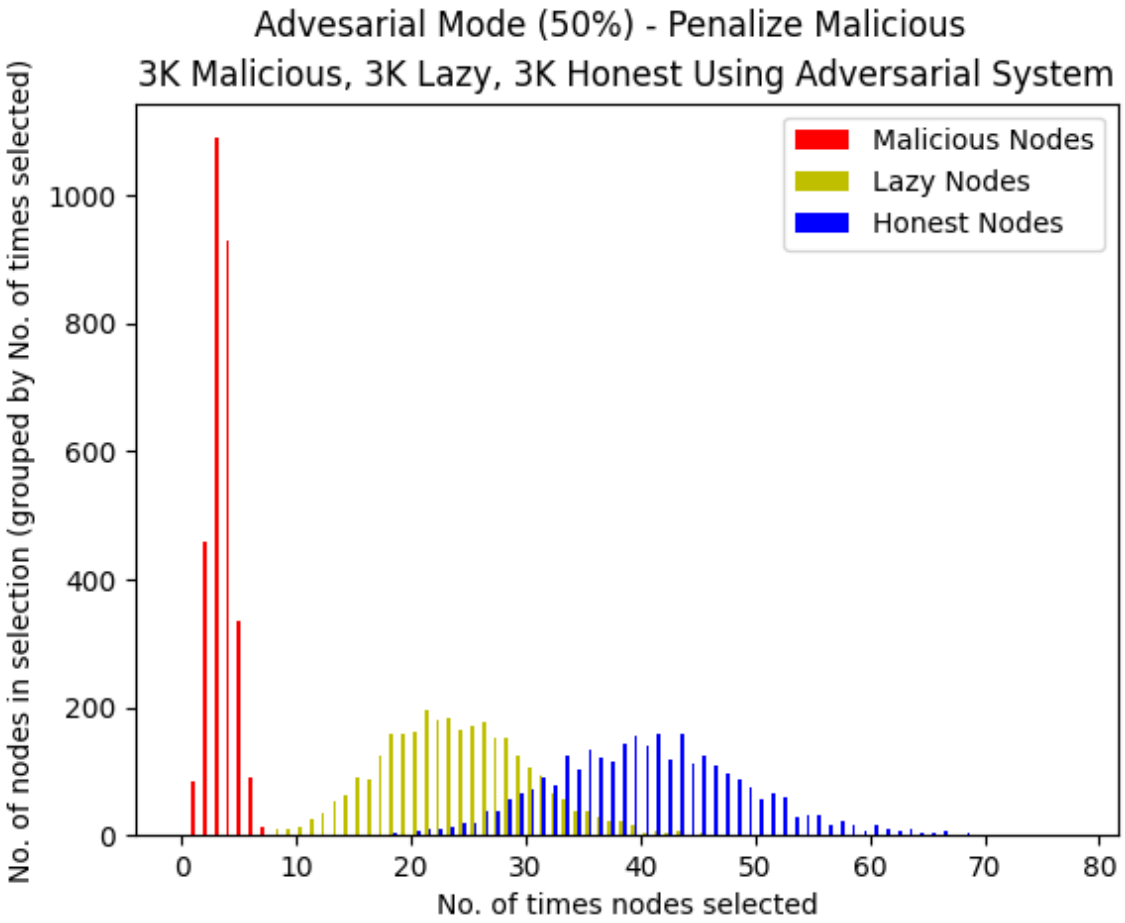

FIGURE 6.5: Node selection rates with adversarial system

A simulator was created to test selection rates for honest, malicious, and lazy nodes over time. Additionally, the number of malicious blocks that could potentially be generated based on the number of type of nodes in each simulated committee was also collected. For the simulation we identified three primary actors within the network with their own respective goals during testing. The three actors are honest nodes, malicious nodes, and lazy nodes.

Honest nodes contribute the network in a positive way. They evaluate all blocks properly and provide a correct response back to accept the block as valid or to reject the block

and provide the correct reason code for rejecting the block. Reasons for rejecting a block could include, but are not limited to, the inclusion of an incorrectly formatted transaction, transactions with insufficient funds, blocks with incorrect height values and many more.

Malicious nodes work together for a common goal to disrupt the network. Malicious nodes act as a collective rather than individual nodes with their own respective goals. If each malicious node worked independently, the chances of malicious nodes voting in favor of malicious nodes are reduced and therefore malicious nodes in this network are colluding to increase the risk they present to the network. For testing, malicious nodes will vote to reject blocks that are not proposed by other malicious nodes and will vote to accept all blocks proposed by malicious nodes. Additionally, blocks proposed by a malicious node are assumed to be invalid and would not be accepted by honest nodes participating in the committee.

Lazy nodes do not care about health of the network, rather only care about earning as much reward as possible with the least computational effort. Lazy nodes work on the assumption that the protocols in place minimize the chance of malicious nodes being present and the data proposed is from an honest node. With this assumption, lazy nodes do not perform validation and accept proposed blocks by default. For testing, lazy nodes will vote to accept all blocks regardless of the proposer node being honest, malicious or another lazy node. Additionally, the assumption was made that while malicious nodes will vote against any node that is not a malicious producer, they will not conduct validation of the block data and thus will not reject with the correct reason code, resulting in the nodes still receiving a reputation score penalty even though they vote to reject the block.

The simulator system focused on gathering data for several distinct scenarios for comparison purposes. Each Scenario had a shared baseline with 9,000 total simulated nodes, used a committee size of 101 nodes and included 10 blocks per epoch before a new committee was selected. A total of 3,000 epochs or 30,000 blocks were simulated. The first 1,000 epochs included a set of bootstrap nodes to ensure that the first 1,000 epochs were all honest to generate reputation score data for the system. What was different between the scenarios included the proportions of honest, malicious, and lazy nodes within the 9,000 simulated nodes and whether a simple penalty system was used, no penalty system was used or if the adversarial system was used. If the adversarial system was used, additional scenarios were also conducted that evaluated the rate at which adversarial blocks were generated in lieu of regular blocks. The honest/malicious/lazy node breakdowns used are highlighted below:

- 6,000 honest/1,500 malicious/1,500 lazy
- 6,000 honest/3,000 malicious/0 lazy
- 4,500 honest/2,250 malicious/2,250 lazy
- 3,600 honest/2,700 malicious/2,700 lazy

Two main observations were derived from analysis of the experiment data. First, lazy nodes pose a significant risk to the network when malicious nodes are present regardless of the number of the malicious nodes. Secondly, adversarial consensus appears effective at reducing the impact of lazy nodes on the network and appears to create resiliency network even when only 40% of the nodes in the network are honest.

The presence of lazy nodes on the network can amplify the impact of malicious nodes present in the network even when malicious nodes are impacted by negative scores effecting their selection rates. However, it is important to note that this is contingent upon the combination of lazy and malicious nodes making up approximately half of the nodes in the network. Malicious blocks were still generated when only 200 malicious nodes were included with 4,400 lazy nodes. When no penalty system was used 292 malicious blocks were created, however, even 42 malicious blocks were still created when the penalty system was used when the adversarial system was turned off. The impact of lazy nodes being present produced similar results to scenarios where no lazy nodes were present but there was a high number of malicious nodes. The same rate of malicious blocks was generated when 2/3 of the nodes were honest and 1/3 were either all malicious or split evenly between lazy and malicious nodes. The number of malicious blocks produced in experiments can be seen in Table 6.1.

Experimentation has shown that the adversarial consensus mechanism is able to further reduce the rate of malicious block creation even if lazy nodes are present. Malicious block creation when lazy nodes are present occurs when the block producer is malicious and the number of malicious and lazy nodes present in the committee is greater then the number of honest nodes. The adversarial approach reduced selection rates for both malicious and lazy nodes. Higher adversarial rates resulted in a larger impact on the selection of lazy nodes at the expense of reducing the block production rate. Adversarial blocks result in a change to reputation scores but do not update the account state or processes transactions in the block. Each transaction in an adversarial block needs to be included in another non-adversarial

TABLE 6.1: Number of malicious blocks created with and without penalty system

Honest_Malicious_Lazy	With Penalty	Without Penalty
6000_1500_1500	0	6
6000_3000_0	0	5
3000_3000_3000	964	6702
3600_2700_2700	175	5947
4400_200_4400	42	292

TABLE 6.2: Comparison of experiments with varying rates of adversarial blocks

Honest_Malicious_Lazy	0% Rate	20% Rate	30% Rate	40% Rate
3600_2700_2700	175	48	7	0
4500_2250_2250	49	5	0	0
4400_200_4400	42	30	0	0

block for processing. If a 30% adversarial block rate was used, the network remained resilient against malicious block creation even when honest nodes only consisted of 50% of all nodes. By increasing the adversarial rate to 40% the network was stable even when only 40% of the network consisted of honest nodes. The impact of the adversarial approach is also evident by examining a scenario with 49% honest and lazy nodes and 2% malicious nodes. In this scenario, without the adversarial approach, 42 malicious blocks were able to be created, and 30 were created when the adversarial option system was 20%. The implication is that adversarial block rates may need to be over 20% or longer bootstrapping time may be needed to apply sufficient penalties to lazy nodes to impact future selection rates. Table 6.2 displays the results of adversarial consensus scenarios.

CHAPTER 7

CONCLUSION

This dissertation contributes to the field of blockchain technology by presenting an alternative architecture for blockchain technology that aims to reduce some of the drawbacks of existing approaches. Through research efforts, we have shown how the framework presented provides a method to reliably implement a reputation-based committee member selection mechanism that is fair and centralization resistant. The selected members of the committee participate in an active analysis of the behavior of other members to penalize dishonest behavior, resulting in a higher population of honest nodes being selected compared to dishonest nodes. Additionally, the mechanism is resistant to Sybil attacks by malicious actors by applying a cost, in the form of work execution, prior to being able to apply for consideration as a committee member. Multiple concepts from existing blockchain and distributed computing implementations were analyzed and incorporated into the framework in such a way that capitalized on the benefits they provide, yet minimize the limitations, creating a novel approach that has not been previously considered. The integration between the Interplanetary File System (IPFS) network and the blockchain architecture created a system that expands the distributed storage of the blockchain data and further enhances data integrity across the network. The use of the IPFS content identifiers, ensures that data will not change and minimizes the size of transactions and blocks by replacing content with references to the hashes, which can be used to retrieve the content. Additionally, exposure to the IPFS

network, allows external storage of blockchain data beyond blockchain nodes while simultaneously allowing blockchain nodes to selectively prune data without impacting the ability of data to be retrieved. Finally, the architecture showed that IPFS could be used as a mechanism to enable distributed computation of arbitrary code by providing a mechanism for the recreation of all files required for execution, in a way that generates verifiable deterministic results. All the various components of the architecture work together to create a novel design for blockchain architecture that provides many contributions to several areas of research. A detailed summary of the contributions made as part of this dissertation are described in the following sections. Additionally, we wrap up also identify areas where the work could still continue as areas for improvement or additional topics to be researched.

7.1 A BLOCKCHAIN DESIGNED WITH IPFS INTEGRATION

A blockchain is a network consisting of independently operated nodes maintaining identical data and records in a decentralized way. While all the nodes maintain a synchronized set of data, each node is responsible for the storage of all the data locally creating duplication of data. If a node is missing data from its respective local storage system, it will have an incomplete picture of the network state which can lead to faulty behavior. IPFS is a system that acts as a distributed storage network, where data is stored based on the hash of the content, rather than the physical location where the server exists. Using distribute hash tables, if a user knows the respective content identifier hash for a file, and at least one copy of the file exists somewhere on the IPFS network, the file will be able to be retrieved. By integrating IPFS with the blockchain architecture, nodes that validate blocks and transactions, will store the files locally, as well as on IPFS. When users request the data, the data will be

propagated to additional nodes, which may exist outside the IPFS network, further expanding the availability of the data. Additionally, since only one instance of a file is required to be stored on IPFS for it to be retrievable, it is possible for blockchain nodes to prune some of the data to save local storage space without introducing a risk of a node becoming faulty. Up to this point, although the two technologies are similar, no other research has been identified that integrate the two technologies in this way, creating a novel solution for blockchain data storage.

7.2 A COMMITTEE-BASED CONSENSUS MECHANISM WITH REPUTATION-BASED BIAS FOR SELECTION

Consensus mechanisms are a core component of a blockchain and are what enable the nodes in the network to maintain synchronized data. In systems where a single node is chosen at a time, it is possible that the node chosen may have malicious intent or may attempt to censor some data on the network. Using a committee-based approach, risks associated with single nodes is reduced, since the data proposed must be validated by multiple independently operated nodes within the committee to be accepted. However, committee-based systems are only reliable if a majority of the members of the committee are honest. Blockchain nodes will typically evaluate block data that is submitted across the network and if the data is invalid, it is ignored. However, this approach prevents nodes from recording the behavior of nodes that submitting invalid data.

Our work demonstrated a way to record how nodes vote by separating the raw block data from voting data. We showed how three integrated block types could be used to record transaction data, votes from participating nodes and changes to the blockchain state in a way that has not been used previously. Committee nodes that vote within a majority are rewarded

with a boost to their respective reputation score while those that vote in the minority are penalized. The selection algorithm considers the score in such a way the selection of honest nodes with higher scores occurs more often than potentially malicious nodes with lower scores. While this approach was developed specifically for the FAWAC architecture, the concepts can be modified and applied to the broader range of committee-based network agreement outside of blockchain networks.

7.3 IDENTIFICATION OF THE LAZY AGREEMENT PROBLEM

The reputation-based committee selection process demonstrated by our research showed that over time, the selection of malicious nodes will decrease for committees. However, the algorithm does not completely prevent a malicious node from potentially being selected to be part of a committee. With the understanding that most of the nodes are likely to be honest, a new problem was identified during the process of our research. A new vulnerability could be created if a node decides that due to the higher probability of the committee being honest, the node could bypass validation and vote to accept the block. We identified this behavior as “Lazy Agreement" and prior to our research, it has not been highlighted as a potential security risk.

While lazy agreement does not introduce any risk when an honest node proposes block data, if a malicious node proposes an invalid block, it increase the chances of the malicious block being accepted. Similar to the committee selection mechanism, the lazy agreement problem could also be applied to areas outside of this specific blockchain design.

7.4 A SOLUTION TO THE LAZY AGREEMENT PROBLEM

While our research did identify an issue with a pure reputation-based committee member selection process with the lazy agreement problem, we also designed a countermeasure to reduce the impact of the problem. The solution involves allowing block proposers to purposefully submit bad data to detect nodes that agree to accept without performing validation and penalizing those nodes as appropriate. As the identification of the lazy agreement problem was a new contribution provided by our work, it is also understandable that no solution to the problem existed prior to our research creating an additional contribution to the field of security for committee-based consensus.

7.5 A METHOD FOR VERIFIABLE EXECUTION OF CODE ACROSS DISTRIBUTED NODES

Distributed computation is an area of research that enables participating nodes to submit work items to an external system to be executed on the original node's behalf. Our research explored an approach to distributed computing that uses IPFS nodes as a shared back-end data source that can be used to upload and retrieve data without relying on a centralized authority that had not been previously explored. We showed that by using python code, configuration files and data sources, key data could be uploaded to IPFS to generate content identifies that could be used to retrieve the data by remote systems for execution. Additionally, as nodes store the results from executing the code on IPFS, a content identifier is created which will be identical for all nodes which honestly executed the code and generate the same solution.

Comparison of nodes which produce identical solutions is used as a metric to determine the correctness of the solution without having to know the correct solution ahead of time. Although the solution we developed was designed specifically as a mechanism to prevent Sybil attacks against the network, the concepts can be adapted to be applicable to a wide range of research related to distributed computation beyond the scope of our initial work.

7.6 LIMITATIONS AND FUTURE OPPORTUNITIES

Although the FAWAC architecture introduces several improvements to existing blockchain architectures, we recognize that areas still exist where the system can be further optimized to enable it to operate more efficiently at larger scales. While the items identified here do not preclude the system from operating, the implementation of the improvements could further enhance the usability of the system and lead to wider adoption.

7.6.1 Enhanced User Interfaces

The current iteration of the FAWAC architecture was designed for proof of concept testing and operation within a small controlled community of developers and researchers. The interface to interact with the system were primarily designed to support command line interface interaction and did not focus on the generation of user-friendly usability. Transactions are submitted through command line prompts to collect user input and properly format the transactions before submitting to nodes. Similarly, while data from the blocks can be queried, results a primarily in the form of IPFS content identifiers. The identifiers need to be passed to an IPFS Node to extract the data, and in situations where the content includes

additional IPFS content identifier references, the process needs to be repeated. The development of a user interface that simplifies the process of submitting and reading data would enhance the usability of the system and potentially lead to greater adoption.

7.6.2 Improved Storage of State Data

The Transaction Enclave within the FAWAC architecture uses a simple dictionary to record the state of user accounts. While the size of each individual record can be less than 100 bytes, when the system incorporates thousands of participating accounts, the size of the state file can grow to several megabytes in size. Since each change is saved to IPFS, a new version is created, which could take up additional space for every block created if the history of every state is maintained. While pruning can be used to reduce the impact on storage requirements, more efficient storage mechanisms can also be used to reduce the size of the account state. Through the use of prefix trees, similar content identifiers can be combined to compress similar portions of account state keys, saving bytes as more accounts are added to the network.

7.6.3 Fallback Work Item Generation

A key component of the FAWAC framework is the advanced work component where work items are generated, selected and executed. The completion of the work is a requirement for nodes to be considered as part of the transaction enclave committee. However, if there is no available work being generated, the system may be at risk of stalling once all nodes that have completed work have already participated in committees and are unable to complete additional work. One possible solution is to have a mechanism to monitor the availability

of work items and to begin to generate “useless” work as a backup to prevent the system from stalling. While the approach is not optimal, it would ensure a situation will not be encountered where a committee cannot be selected due to a lack of completed work.

BIBLIOGRAPHY

[1] Adams, S. C. and Zheng, Y. (2022). A framework using useful work for transient committee selections in blockchain consensus. In *2022 International Conference on IoT and Blockchain Technology (ICIBT)*, pages 1–6.

[2] Adams, S. C. and Zheng, Y. (2023a). An adversarial approach to mitigating lazy voting threats to committee-based consensus. In *2023 International Conference on Artificial Intelligence, Blockchain, Cloud Computing, and Data Analytics (ICoABCD)*, pages 71–76.

[3] Adams, S. C. and Zheng, Y. (2023b). A blockchain design supporting verifiable reputation-based selection of committee members and ipfs for storage. In *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*, pages 200–208.

[4] Adleman, L. and Rivest, R. (1978). The use of public key cryptography in communication system design. *IEEE Communications Society Magazine*, 16(6):20–23.

[5] Andrianopoulos, A. (2017). *Mastering Bitcoin: Programming the Open Blockchain 2nd Edition*. O'Reilly Media.

[6] Andrianopoulos, A. (2018). *Mastering Ethereum*.

[7] Anthal, J., Choudhary, S., and Shettiyar, R. (2023). Decentralizing file sharing: The potential of blockchain and ipfs. In *2023 International Conference on Advancement in Computation & Computer Technologies (InCACCT)*, pages 773–777.

[8] Back, A. (2002). Hashcash - a denial of service counter-measure.

[9] Baldominos, A. and Saez, Y. (2019). Coin.ai: A proof-of-useful-work scheme for blockchain-based distributed deep learning. *Entropy*, 21(8).

[10] Bano, S., Sonnino, A., Bassam, M. A., Azouvi, S., McCorry, P., Meiklejohn, S., and Danezis, G. (2017). Consensus in the age of blockchains. *CoRR*, abs/1711.03936.

[11] Baumgart, I. and Mies, S. (2007). S/kademlia: A practicable approach towards secure key-based routing. In *2007 International Conference on Parallel and Distributed Systems*, pages 1–8.

[12] Bazm, M.-M., Lacoste, M., Südholt, M., and Menaud, J.-M. (2018). Secure distributed computing on untrusted fog infrastructures using trusted linux containers. In *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 239–242.

[13] Bedford Taylor, M. (2017). The evolution of bitcoin hardware. *Computer*, 50(9):58–66.

[14] Benet, J. (2014). Ipfs - content addressed, versioned, p2p file system.

[15] Benet, J. and Greco, N. (2018). Filecoin: A decentralized storage network. *Protoc. Labs*, pages 1–36.

[16] Buterin, V. (2015). A next generation smart contract and decentralized applilcation platform. Technical report, etherum.org.

[17] Buterin, V. and Griffith, V. (2019). Casper the friendly finality gadget.

[18] Buterin, V., Hernandez, D., Kamphefner, T., Pham, K., Qiao, Z., Ryan, D., Sin, J., Wang, Y., and Zhang, Y. X. (2020). Combining ghost and casper.

[19] Casanova, H. (2002). Distributed computing research issues in grid computing. *SIGACT News*, 33(3):50–70.

[20] Castro, M. and Liskov, B. (2002). Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461.

[21] Chase, B. and MacBrough, E. (2018). Analysis of the XRP ledger consensus protocol. *CoRR*, abs/1802.07242.

[22] Chaudhry, N. and Yousaf, M. M. (2018). Consensus algorithms in blockchain: Comparative analysis, challenges and opportunities. In *2018 12th International Conference on Open Source Systems and Technologies (ICOSST)*, pages 54–63.

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

[24] Costello, C., Fournet, C., Howell, J., Kohlweiss, M., Kreuter, B., Naehrig, M., Parno, B., and Zahur, S. (2015). Geppetto: Versatile verifiable computation. In *2015 IEEE Symposium on Security and Privacy*, pages 253–270.

[25] Crain, T., Gramoli, V., Larrea, M., and Raynal, M. (2018). Dbft: Efficient leaderless byzantine consensus and its application to blockchains. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–8.

[26] Davlyatov, S. (2023). Smart-city ecosystem using block-chain technology. In *2023 3rd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*, pages 1077–1080.

[27] de Brito Gonçalves, J. P., Spelta, G., da Silva Villaça, R., and Gomes, R. L. (2022). Iot data storage on a blockchain using smart contracts and ipfs. In *2022 IEEE International Conference on Blockchain (Blockchain)*, pages 508–511.

[28] de Camargo, R. Y. and Kon, F. (2006). Distributed data storage for opportunistic grids. In *Proceedings of the 3rd International Middleware Doctoral Symposium*, MDS '06, page 3, New York, NY, USA. Association for Computing Machinery.

[29] Deirmentzoglou, E., Papakyriakopoulos, G., and Patsakis, C. (2019). A survey on long-range attacks for proof of stake protocols. *IEEE Access*, 7:28712–28725.

[30] Diffie, W. and Hellman, M. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654.

[31] Dillon, T., Wu, C., and Chang, E. (2010). Cloud computing: Issues and challenges. In *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pages 27–33.

[32] Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., and Stehlé, D. (2018). Crystals-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1):238–268.

[33] Eyal, I. and Gun Sirer, E. (2013). Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security: 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*, volume 8437.

[34] Foster, I., Zhao, Y., Raicu, I., and Lu, S. (2008). Cloud computing and grid computing 360-degree compared. In *2008 Grid Computing Environments Workshop*, pages 1–10.

[35] Fournet, C., Keller, C., and Laporte, V. (2016). A certified compiler for verifiable computing. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 268–280.

[36] Games, M. (1970). The fantastic combinations of john conway's new solitaire game "life" by martin gardner. *Scientific American*, 223:120–123.

[37] Gao, W., Hatcher, W. G., and Yu, W. (2018). A survey of blockchain: Techniques, applications, and challenges. In *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–11.

[38] Gilad, Y., Hemo, R., Micali, S., Vlachos, G., and Zeldovich, N. (2017). Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 51–68, New York, NY, USA. ACM.

[39] Grigg, I. (2017). Eos: An introduction. Technical report, Block.IO.

[40] Hsiao, H., Liao, H., and Huang, C. (2009). Resolving the topology mismatch problem in unstructured peer-to-peer networks. *IEEE Transactions on Parallel and Distributed Systems*, 20(11):1668–1681.

[41] Hyperledger (2017a). Hyperledger fabric.

[42] Hyperledger (2017b). Hyperledger sawtooth.

[43] Johnson, D., Menezes, A., and Vanstone, S. (2001). The elliptic curve digital signature algorithm (ecdsa). *International Journal of Information Security*, 1(1):36–63.

[44] Kashef, R. and Niranjan, A. (2017). Handling large-scale data using two-tier hierarchical super-peer p2p network. In *Proceedings of the International Conference on Big Data and Internet of Thing*, BDIOT '17, page 52–56, New York, NY, USA. Association for Computing Machinery.

[45] Konapure, R. R. and Nawale, S. D. (2022). Smart contract system architecture for pharma supply chain. In *2022 International Conference on IoT and Blockchain Technology (ICIBT)*, pages 1–5.

[46] Kwang-Hui Lee (1994). A distributed network management system. In *1994 IEEE GLOBECOM. Communications: The Global Bridge*, pages 548–552 vol.1.

[47] Lakshmish Ramaswamy, Gedik, B., and Liu, L. (2005). A distributed approach to node clustering in decentralized peer-to-peer networks. *IEEE Transactions on Parallel and Distributed Systems*, 16(9):814–829.

[48] Larimer, D. (2017). Eos.io technical white paper. Technical report, block.one.

[49] Li, X., Jiang, P., Chen, T., Luo, X., and Wen, Q. (2017). A survey on the security of blockchain systems. *Future Generation Computer Systems*.

[50] Lihu, A., Du, J., Barjaktarevic, I., Gerzanics, P., and Harvilla, M. (2020). A proof of useful work for artificial intelligence on the blockchain.

[51] Liu, J., Li, W., Karame, G. O., and Asokan, N. (2019). Scalable byzantine consensus via hardware-assisted secret sharing. *IEEE Transactions on Computers*, 68(1):139 – 151.

[52] Ma, Y., Fu, Y., Liu, L., Du, Z., JingMa, and Sun, Y. (2022). A smart contract approach to access control based on distributed identities and roles. In *2022 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS)*, pages 677–680.

[53] Maurer, A. and Tixeuil, S. (2014). Self-stabilizing byzantine broadcast. In *2014 IEEE 33rd International Symposium on Reliable Distributed Systems*, pages 152–160.

[54] Meng, Y., Cao, Z., and Qu, D. (2018). A committee-based byzantine consensus protocol for blockchain. In *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*, pages 1–6.

[55] Merlina, A. (2019). Blockml: A useful proof of work system based on machine learning tasks. In *Proceedings of the 20th International Middleware Conference Doctoral Symposium*, Middleware '19, page 6–8, New York, NY, USA. Association for Computing Machinery.

[56] Mohaisen, A., Tran, H., Chandra, A., and Kim, Y. (2014). Trustworthy distributed computing on social networks. *IEEE Transactions on Services Computing*, 7(3):333–345.

[57] Montes, J. M., Ramirez, C. E., Gutierrez, M. C., and Larios, V. M. (2019). Smart contracts for supply chain applicable to smart cities daily operations. In *2019 IEEE International Smart Cities Conference (ISC2)*, pages 565–570.

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

[59] Nakamoto, S. (2009). Bitcoin: A peer-to-peer electronic cash system. Technical report, bitcoin.org.

[60] NGUYEN, B. (2017). Exploring applications of blockchain in securing electronic medical records. *Journal of Health Care Law & Policy*, 20(1):99 – 115.

[61] Parno, B., Howell, J., Gentry, C., and Raykova, M. (2013). Pinocchio: Nearly practical verifiable computation. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE. Best Paper Award.

[62] Puri, V., Solanki, V. K., Kataria, A., and Long, C. K. (2023). Blockchain-enabled transparent and trustworthy regulation system for smart cities. In *2023 First International Conference on Cyber Physical Systems, Power Electronics and Electric Vehicles (ICPEEV)*, pages 1–6.

[63] Saraf, C. and Sabadra, S. (2018). Blockchain platforms: A compendium. In *2018 IEEE International Conference on Innovative Research and Development (ICIRD)*, pages 1–6.

[64] Sarmady, S. (2007). A peer-to-peer dictionary using chord dht. *University of Sains Malaysia, Technical Report*.

[65] Sunny, K. and Scott, N. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake.

[66] Turan, F. and Verbauwhede, I. (2019). Propagating trusted execution through mutual attestation. In *Proceedings of the 4th Workshop on System Software for Trusted Execution*, SysTEX '19, New York, NY, USA. Association for Computing Machinery.

[67] Uddin, M. N., Hasnat, A. H. M. A., Nasrin, S., Alam, M. S., and Yousuf, M. A. (2021). Secure file sharing system using blockchain, ipfs and pki technologies. In *2021 5th International Conference on Electrical Information and Communication Technology (EICT)*, pages 1–5.

[68] van Rossum, J. and Lawlor, B. (2018). The blockchain and its potential for science and academic publishing. *Information Services & Use*, 38(1/2):95 – 98.

[69] Wang, J., Hong, Z., Zhang, Y., and Jin, Y. (2018). Enabling security-enhanced attestation with intel sgx for remote terminal and iot. volume 37, pages 88–96.

[70] Wood, G. Ethereum: A secure decentralised generalised transaction ledger eip-150 revision.

[71] Yadav, A. K. (2021). Significance of elliptic curve cryptography in blockchain iot with comparative analysis of rsa algorithm. In *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, pages 256–262.

[72] Yang, J., Jia, Z., Su, R., Wu, X., and Qin, J. (2022). Improved fault-tolerant consensus based on the pbft algorithm. *IEEE Access*, 10:30274–30283.

[73] Yuan, Y. and Wang, F.-Y. (2018). Blockchain and cryptocurrencies: Model, techniques, and applications. *IEEE Transactions on Systems, Man & Cybernetics. Systems*, 48(9):1421 – 1428.

ABOUT THE AUTHOR

Shawn C. Adams was born in Hazleton, Pennsylvania, USA in 1985. He received the B.S in Computer Science from Bloomsburg University of Pennsylvania, in 2007 and the M.S. in Information Technology Management from Central Michigan University in 2012. He received a second M.S. in Military Theory and International Relations while attending Air Command and Staff College at the Air University, Maxwell Air Force Base, Alabama in 2023.

From 2007 to 2017, he worked as a Cyberspace Operations Officer for the United States Air Force before separating to pursue academic and business interests. Since 2017, he has worked as a Senior Consultant for CGI in the Birmingham AL office.

His research areas are related to blockchain and distributed computing. He was the first author of five academic papers that were selected to be included and presented at conferences hosted by IEEE. His primary research effort is related to the Federated Advanced Work Adversarial Consensus (FAWAC) blockchain architecture which is designed to provide efficient block production and fair selection of committee-based validation nodes.

His major published papers include:

- S. C. Adams and Y. Zheng, "An Adversarial Approach to Mitigating Lazy Voting Threats to Committee-Based Consensus," 2023 International Conference on Artificial Intelligence, Blockchain, Cloud Computing, and Data Analytics (ICoABCD), Denpasar, Indonesia, 2023, pp. 71-76, doi: 10.1109/ICoABCD59879.2023.10390964.
- S. C. Adams and Y. Zheng, "A Blockchain Design Supporting Verifiable Reputation-based Selection of Committee Members and IPFS for Storage," 2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C), L'Aquila, Italy, 2023, pp. 200-208, doi: 10.1109/ICSA-C57050.2023.00053.
- S. C. Adams and Y. Zheng, "A Framework Using Useful Work for Transient Committee Selections in Blockchain Consensus," 2022 International Conference on IoT and Blockchain Technology (ICIBT), Ranchi, India, 2022, pp. 1-6, doi: 10.1109/ICIBT52874.2022.9807709.
- S. C. Adams and Y. Zheng, "FAWAC: An Attack-Resistant, Multi-Enclave Distributed Ledger Architecture," 2021 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS), Paris, France, 2021, pp. 35-36, doi: 10.1109/BRAINS52497.2021.9569790.
- S. C. Adams and Y. Zheng, "*A Blockchain Smart Contract Framework Using Interpreted Programming Languages and Decentralized Storage"*, IEEE Southeast Conference March 2024 (pending publishing)