PARAMETRIZED ANALYSIS OF THE MOST WIDELY USED CONSENSUS PROTOCOLS IN BLOCKCHAIN

By

Muhammad Ehsan Ullah Khan

A thesis submitted to the faculty of Department of Information Security, Military College of Signals, National University of Sciences and Technology, Islamabad, Pakistan, in partial fulfillment of the requirements for the degree of MS in Information Security

Sep 2019

ABSTRACT

Blockchain has caught the attention of both researchers and industrialists because of its widespread growth in the last 10 years. What was started with decentralized immutable ledger has now paved its way to decentralized, open access and self-organizing networks and journey has just begun. These characteristics like decentralization, immutability, open access, liveliness and trust are achieved through consensus protocol employed by specific Blockchain. A consensus protocol allows a group of separate nodes to distribute the right to update the system according to specific rules, amongst a set of participants, and in a secure way. A plethora of such consensus protocols have been proposed since last few years. However, not all those protocols are as robust as others and existing systems in the same domains, few offer higher security guarantees than others and have their unique performance characteristics. One mechanism will not fit every envisaged application and use case of blockchain. This mandates that such protocols be compared for their security and performance parameters to gain a comprehensive overview of their strengths and weaknesses. This research covers a comparison of nine consensus protocols designed to operate in private, consortium and public blockchain settings belonging to different eras of protocols design and enhancements. Current research provides an overview of generic functioning of blockchain and a summary and guideline for application developers to choose a consensus protocol for their applications that best match their requirements and quick reference for those protocols for the academia — doing so it also provides recent directions in consensus designing landscape.

Keywords

Blockchain Consensus, Distributed System Consensus, Blockchain System, Blockchain Design, Asymmetric Cryptography, Hashing, Bitcoin, Nakamoto Consensus, Bitcoin-NG, Proof of stake, Ouroboros, Proas, PBFT

DEDICATION

This thesis is dedicated to

MY FAMILY, FRIENDS AND TEACHERS

for their love, endless support and encouragement

ACKNOWLEDGEMENTS

I am grateful to God Almighty who has bestowed me with the strength and the passion to accomplish this thesis and I am thankful to Him for His mercy and benevolence. Without his consent I could not have indulged myself in this task.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

Blockchain has been heralded as a disruptive innovation that will impact every sphere of our lives by providing a trustworthy service to people and businesses that may or may not fully trust each other. It is promised to revolutionize the business and the industry. Blockchain based crypto currencies and initial coin offerings (ICOs) crossed over $225 billion in the global market in 2019 [19] and expected huge growth in coming years due to its key characteristics of integrity, resilience and transparency. Blockchains are instances of decentralized, immutable replicated state machines providing a transparent and tamper-evident log. Being immutable, data on a blockchain can not be tampered. Any network participant node (client) can access it by joining the chain network and verify its correctness either by downloading the entire chain or through other full nodes operating in the reference blockchain network. To create a transaction, a node generates a transaction and submits it to the referenced blockchain network. Full nodes on the blockchain will process that transaction. If the transaction meets the rules specified by the chain, state of the replicated state machine will change, and the transaction will be added to the chain in a block. A crucial significance of this architecture is *disintermediation* operating under a partially synchronous environment. Based on the operational environment of the blockchain (private, consortium or public), trusted, semi-trusted or untrusted parties directly and transparently interact to update the system state.In case of public and consortium blockchains, multiple parties can update the system, and all those parties can do that at the same time. That is when a node receives two transactions generated at the same time, it has to agree on ordering for those transactions to process further. Thus, the problem of total ordering [32] and consistency of data sets may arise in such systems. If we further boil down the total ordering problem, we need to achieve two properties for each transaction to process it. These properties are *liveness* and *safety* [50]. Liveness property ensures that every request from a honest node eventually gets processed where as the safety property insures that if one of the honest node on the network accepts a transaction as correct, then all other honest nodes on the network also accepts that

transaction as a valid transaction.

These plausible characteristics of total ordering and immutability in a distributed system are primarily a function of distributed consensus and owe a rapidly developing landscape. Especially after academia and industry discussed the limitations of Bitcoin, a plethora of new proposals emerged aimed at enhancing transaction throughput and maintaining security at least that of Bitcoin. However, only a few of these proposals got implemented and mostly sacrificed one property for the other or made unrealistic assumptions. Right after the Bitcoin, new proposals were mostly suggested for the public blockchains to operate in open settings where anyone can join the network and can take an active part in the decision making for the next blocks to be added to the chain. This required those to scale to the network. To achieve this network scalability, similar to Bitcoin, most of the protocols sacrificed the performance that is reduced the transaction throughput. Though it was better than the Bitcoin, these reduced the complexity of the cryptographic puzzle and reduced the security. Also, these protocols have to work in a partially synchronised environment like the internet. Thus to ensure that all the transaction finally gets added to the chain, these introduced high transaction latency. On the other hand, blockchain proposals for closed settings like Hyperledger Fabric use the classical Practical Byzantine Fault Tolerance consensus protocol. It offers very high transaction throughput but limited network scalability and security. Latest trends on consensus landscape have shown proposals those either propose a new design like proof of stake, or modify existing protocol without compromising security like Bitcoin-Ng or layered architecture for consensus using both proof of X and classical byzantine fault tolerant techniques or using shards with proof of X protocols to enhance the transaction throughput.

Related Work Decentralized system are in use sine early 70's. If all the nodes in the system are honest and working correctly, then the entire network is in consistent state. But problem occurs, if a nodes become faulty or an adversary joins the network. This node may hold tampered data and propagate the same to others. This will lead entire system to a non-consistent state and data has lasted out its credibility. In literature, this problem is known as Byzantine General's problem [72]. Few solutions were proposed to cater for this problem. In literature this class of protocols is termed as classical Byzantine Fault Tolerance (BFT) protocols[94]. However, all such solutions failed to achieve trust in the decentralized systems.

It was until 2008, when Satoshi Nakamoto proposed a cryptocurrency titled ”Bitcoin” [82] based on decentralized machines. This was the first time that anyone proposed a system that was entirely constituent of non-trust systems without any trusted third party. And how the system achieved trust and all other ideals of decentralized environment (correctness, consistency, termination and total order)? The answer is, he proposed a new consensus protocol for the decentralized system called ”Proof of Work” (PoW).
Bitcoin was a mega success. This lead to many other cryptocurrencies like Bitcoin Cash, LiteCoin and hundreds of others generically known as Alt-Coins and decentralized application environments like Ethereum [59] and Hyperledger [24]. These all relied on mechanisms and consensus protocol offered by Bitcoin with few modifications to meet the specific needs for the environment those systems work in.

A lot of research is taking place in making consensus more robust, secure and environment friendly since last three years. However, no significant effort has been made to systematically compare the proposed / in use protocols with reference to some specific parameters as mentioned in the aim. Most of the existing work that deals with consensus is either focuses blockchain use cases or some single aspect of the consensus e.g their effectiveness in permissioned or permission less environment or some specific domain e.g in IoT. However, few comparisons are also available like ”Enterprise Blockchain Platforms - A Comparison” [17]. This article discusses the requirements of a good blockchain and compares Ethereum, Corda and Hyperledger Fabric very generically for privacy issues and scalability capabilities. Article titled ”A Comparison of Blockchain Platforms - Competitors” [20] describes the relationship between scalability, security and decentralization. Author talks about Proof of work and Proof of Stake with premises that only two out of security, scalability and decentralization can be achieved and not all three. A good comparison also has been done in a report published by talentica [52]. This report compares the Bitcoin, Ethereum, Multichain and Hyperledger Fabric for capabilities to handle digital currency, ability to perform in permissioned and permission less environments and its ease of adoption. Another good research on consensus is titled ”SoK: Consensus in the Age of Blockchains” [13]. Authors discuss how different consensus protocols achieve finality i.e reach consensus. Few surveys has also been done like in ”An Overview of Blockchain and Consensus Protocols for IoT Networks”

[79] where author discusses whether a particular blockchain is suited for IoT or not. Some recent surveys include [113] which discusses PoW based protocols under a proposed framework and then discusses different PoS protocol methodologies and their strengths, and [107]. This survey extensively studies PoW based protocols, their strengths and weaknesses. It also talks about PoS and Hybrid solutions based on PoW and PoS briefly.

Contribution This research aims at analytically comparing most famous consensus protocols in the blockchain domain against a set of parameters mentioned in 3 and there by hinting the most useful consensus protocols for various applications based on parametrized data for security against adversary, performance using throughput and latency. This does not cover the entire spectrum, but covers the most studied protocols from following different consensus families:

1. *Proof of X* based consensus protocols where probability of being elected as leader to add new block to the blockchain is proportional to total resources invested in the system like Proof of Work. These are designed mostly to work in open settings.

2. *Vote based Classical Consensus* have been designed to operate in closed settings with established identities. These new transactions to the chain, if more than certain number of nodes vote for the said transaction.

3. *Hybrid Consensus protocols* are composed of aforementioned categories and designed mostly to operate in open settings.

A very brief introduction to few other consensus protocols and scalability issues is also given besides enumerating few protocols for the Internet-of-Things devices. This makes it possible for the readers to get a general overview about performance and security of different protocols and lays a foundation for understanding the current and future developments on consensus in the distributed computing specifically blockchains.

Organisation

. This thesis derives its organisation from above paragraph mostly. It starts by describing the functioning of blockchain in a very generic fashion along with its building blocks in chapter 2 and culminates the chapter by showing the importance of consensus in the overall system. In chapter 3, selected parameters have been discussed. Proof of X based protocols have

been discussed and compared in chapter 4. It covers Nakamoto consensus from Bitcoin, Nakamoto consensus from Bitcoin-NG and proof of stake form Ouroboros family. Chapter 5 discusses protocols those reach consensus based on votes by validating nodes. It covers Practical Byzantine Fault Tolerance protocol, BFT-smart and BA* from Algorand. Hybrid of two aforementioned schemes have been discussed in chapter 6. It covers Elastico, Omniledger and ByzCoin. Elastico and Omniledger also benefit sharding. Chapter 7 hints on various protocols best suited for the Internet of Things devices. Few other protocols have been very generically discussed in chapter 8. This chapter also enumerates various open challenges in the consensus spectrum. Finally, thesis work is concluded in chapter9

Chapter 2

What is Blockchain?

This chapter provides a high level view of the blockchain functionality and working. It starts by discovering staple concepts required to understand the blockchain functionality. culminates describing how those different modules fit in to achieve the common goal and It culminates by describing importance of consensus in the entire process.

2.1 Terminologies

2.1.1 Public Key Cryptography

Cryptography is a science of securing digital information from access by unauthorized entities. Two major operations of cryptography are encryption and decryption. Encryption is to protect data by encoding it into some protocol specific format, whereas decryption is the reverse of it and it decodes the encrypted data back to its original form. Metamorphically, encryption is digital equivalent of a protecting something by locking it inside a some type of container whereas decryption is unlocking that container. Encrypted data is called ciphered text and unencrypted data is called plain text. Like in the real word, where keys are required to lock and unlock the locks, keys are also required for encrypting and decrypting data. If same key is used for both the purposes, then it is called symmetric key cryptography and if different inter linked keys are used for encryption and decryption, then it is called asymmetric cryptography or public key cryptography. Public key cryptography uses a pair of complementing (related) keys. One key is derived from the other. Any one key is used for encryption while the other linked key is used for decryption. In fig 2.1, two keys (pair of complementing keys) are depicted labelled Key 1 and Key 2. These have been color coded as blue and red respectively. Once the plain text 'MCS' is encrypted with blue key, it yields cipher text shown in blue box. However, when the same plain text is encrypted using red key, it yields cipher text shown in red box.The two ciphered texts are different as those are a function of the key used for encryption. Now, to decrypt the ciphered text, other key of the pair is need. The key used to encrypt, can not be used for decryption. Which key is to

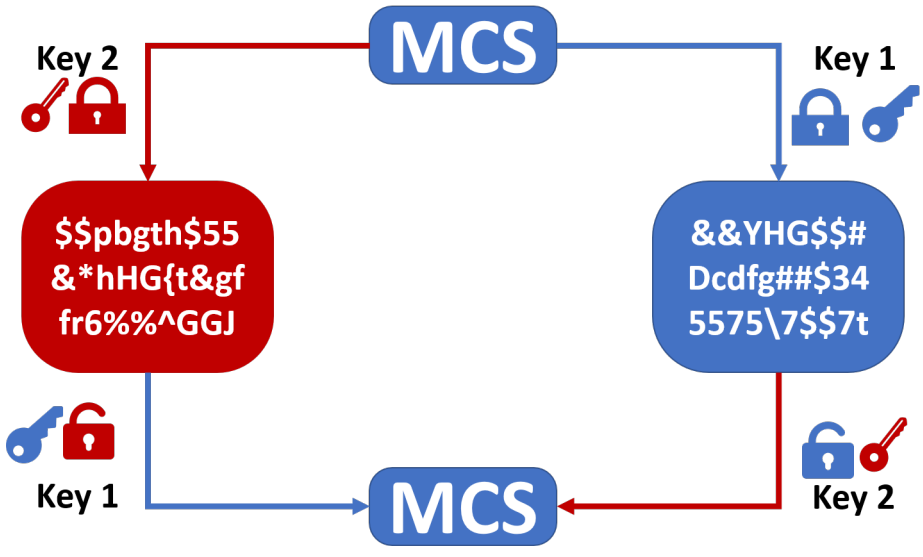


Figure 2.1: Public Key Cryptography working

be used for encryption and which key for the decryption is the owner's choice. Thus in our setting, to decrypt the red box, blue key is needed, whereas to decrypt the blue box, red key is need. A single isolated key is of no use. It can only be used for encryption. This power of asymmetric key distribution allows separating people into groups who are able to encrypt data and other group who can decrypt the data. Based on the role these keys play, these are given names. Typically these are referred as *private* and *public* keys. Public key as indicated by the name, can be given to anyone irrespective of his trustworthiness. However, the private key is to be kept safe and private.

Usage

Public to Private It allows information encryption using public and decryption using private key. As the public key is available to everyone part of the system, any one can send the holder of respective private key an encrypted message. Only the private key holder will be able to decrypt the message. Hence, this use case *allows anyone to send the information to owner of the private key in a secured fashion.* **Private to Public** It allows encryption using private key and decryption using public key. Only the owner of private key will be able to create an encrypted message where as anyone with the respective public key will be able to decrypt it. It is similar to public notice board where only owner of private key publish the message and all the entities with public keys can read it. Hence, this use case is mostly used to prove

authorship. This prove of authorship is mostly referred as *Signatures*. Once the owner of the private key has encrypted the data, it is like it has placed its signatures on that data those can be verified by anyone in the system.

2.1.2 Cryptographic Hash

A hash function transforms any size and kind of data into a fixed size output string. *Cryptographic hash functions* are a special subgroup of hash functions. They meet following properties and are used to digital fingerprint any data:

- *Deterministic*. Given the same data as input, those will always yield same output.
- *One Way Functions*. Function is non-invertible. That is we can only compute output form the input string. From output, input can never be traced back.
- *Collision Resistant*. Probability that two different input will produce same hash is almost negligible. If *Hash*(x_1) = *Hash*(x_2) then x_1 and x_2 should be the same.
- *Pseudorandom*. Output is entirely random. Even a single character change in input will result in about 50% change in output (Avalanche Effect). Fig 2.2 shows SHA1 hash values for different random data. There is a difference of single character in Data and Date. However, it can be seen that hash for Data and Date is completely different.

Data	Hash
Data	e5e429
Date	eb9a4b
Data	e5e429
MS	0f249e
Ms	095fb6

Figure 2.2: Some random data and its truncated SHA1 hash

From a top view, a hash function returns a fixed length hash (random string) for any arbitrary length of data. But a closer look at hash program reveals that it can take a single input of data within limits of an upper bound at a time, for example 256 kilo bits. So, how does it produce hash for an arbitrary length. The answer is to utilize some data arrangement pattern where input to hash function remains within the bounds, yet is able to calculate the hash for entire data. The two interested patterns are:

- *Sequential Hashing*. It works by incrementally updating hash value. It takes hash of the first piece of data in first round and for subsequent rounds, hash of the last round and current data is combined to take the next hash. This way hash for entire data is obtained.

- *Hierarchical Hashing*. Arrange data in an inverted tree fashion with data at leaves. At every step combine two nodes and take there hash. Keep hashing until reach the top, where only a single node is left. This node will represent the hash of the entire data.

2.1.3 Change Sensitive Data Storage

Data can be stored in a change sensitive fashion by storing data together with its hash references, pointing to some other data. The referenced data in turn references some further data, and so on. All the data is linked. If any data or the reference gets forged, all the references from that point onward will break. These broken references serve as an evidence for the forgery. Thus, hash references allow to store data in change sensitive manner.

Merkle Trees

Merkle trees [3] allows data arrangement in an inverted tree pattern. All the data is the leave nodes. In fig 2.3 transactions are the actual data. Merkle trees are useful for grouping distinct data that is all available at the time of processing it and to make it accessible via a single hash reference. To generate the tree we arrange distinctive pieces of data at the leave nodes. Each leave node is processed and its hash reference is created. In fig 2.3 these are labeled as R1 to R4. The hash references created in last process are stored in groups at one level higher than the leave nodes. This group of hash references is hashed again and stored in layer above the current layer (R12, R34). This process continues until only one hash is obtained. This hash represents the entire set of data. If any piece of data or reference is tampered, it will invalidate the whole hash reference. Thus, a broken link is evident that some forgery has been done since data creation. Otherwise, it can be concluded that integrity is maintained that it has not been changed sine creation.

Chain of Hashes

A linked list or chain of linked data is formed once each piece of data holds a hash reference to another piece of data. This type of struct is particularly useful once new data is contin-

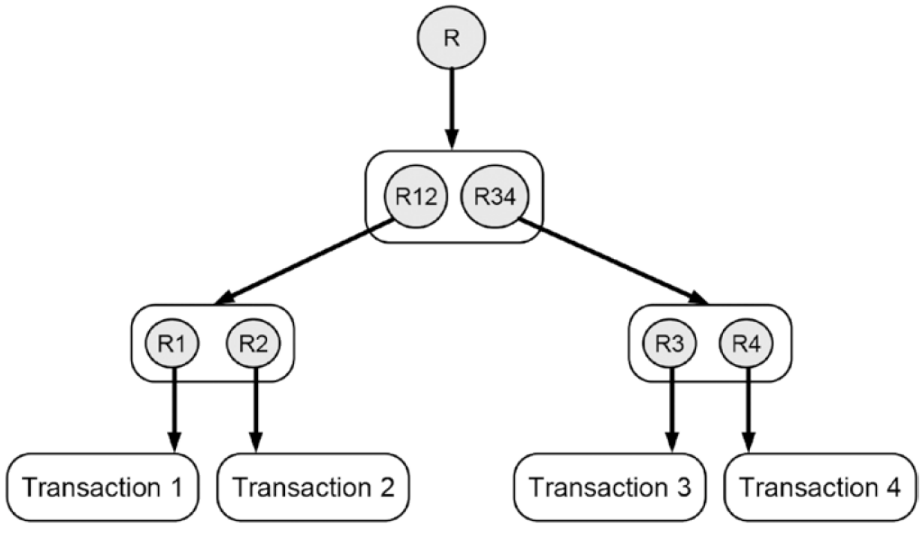


Figure 2.3: Merkle Tree

uously being generated or complete data is not available at a given point. By checking the hash of a particular block, integrity of data till that time can be ascertained. In fig 2.4 it can

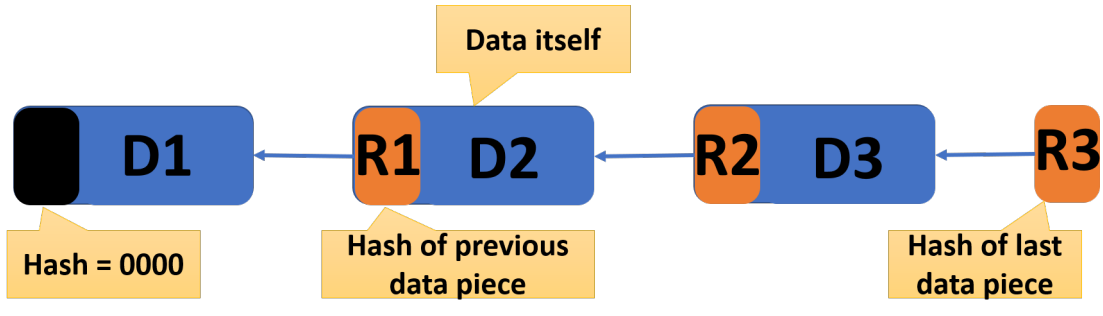


Figure 2.4: Chained Hashes for Tamper Evident Data Storage

be seen that creation of chain starts with data D1 and it does not contain any hash reference. However, once the new data arrives, it is concatenated to hash (R1) of the Data D1. The hash reference R2 references a combined hash of Data D2 and Hash reference R1. Similarly, hash reference R3 refers to combined hash of Data D3 and hash reference R2. The last reference, R3 in this case, is generally known as *head* of the chain and refers to the most recently added piece of data. Only head is needed to traverse the chain in reverse order[36].

2.1.4 Wallets

A wallet is a software application that helps managing an account on blockchain. It is a user's primary interface to underlying blockchain network. A crypto currency wallet does not contain any coins (crypto currency) itself. It holds private keys 2.1.1 for the corresponding

accounts and are used to create and sign transactions on the blockchain network. From a application developer's perspective, wallet is a data structure that is used to store and manage a user's keys [10]. Wallets can be categorized in different classes based on key derivation mechanism and storage. Based on how new keys are derived, wallets are primarily of two types namely non-deterministic wallet (Just a Bunch of Keys) and deterministic wallets. And based on the storage medium used for storing keys, wallets can be classified as Software wallets, Hardware wallets and cold storage wallets. Deterministic key derivation is preferred over non-deterministic and hardware wallets are preferred over software wallets.

2.2 Blockchain - How it functions?

Structure of blocks, the transactions those make up these blocks and how these blocks are chained may differ for different chains but the concept very generically speaking is almost same for most of the existing blockchains. This section describes the general concept how transactions happen, how those are grouped to make up blocks and finally the way those are linked to form the chain of blocks, the blockchain. Bitcoin blockchain being the most famous and the pioneer blockchain will act as reference for this section.

2.2.1 The Block

Transactions

A transaction is a message that *entity A* sends to *entity B*. Entity A sends the transaction/ message to entity B, by taking hash of its entire transaction that it received previously (can be multiple), public key of the entity B, the recipient and then signing this hash using its own private key. As the transaction propagates over the network it will be verified by the other nodes especially the full nodes / miner nodes. If miner nodes find the current transaction correct as per rules of the blockchain in use, it will propagate it further and will keep a copy of it in its global cache sometimes referred as *transaction pool* or *mempool* in a time sensitive fashion. This transaction will travel the network until it reaches the destination node and all the mining nodes.

Block

Once mining nodes have received some transaction they will try to add it in new block. Mostly existing blockchains run in epochs. All the transactions available in an epoch are

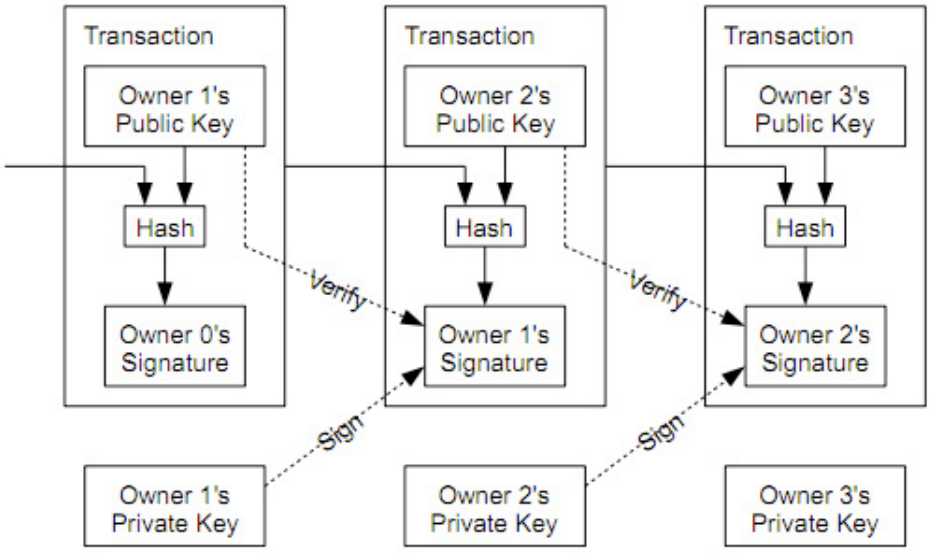


Figure 2.5: Transaction Generation and Validation

added to the new block. Block usually has two portions namely a header and data portion 2.6. Data portion contains all transaction data to be added in current block where as header contains combined hash of all the transactions in the data portion, hash of the last block and any other data as per requirement of the referenced blockchains.

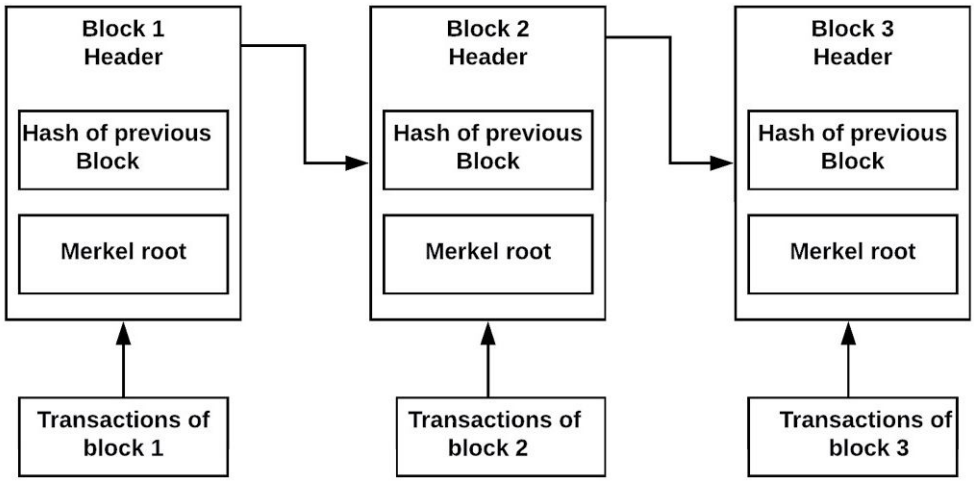


Figure 2.6: Block Structure

2.2.2 The Chain

Once the transactions has been added to the block, a mechanism is required to link the current block to the old block in a chronological order. That is done by placing the hash of last block in the header of the current block. Thus hash of the last block serves as the

link for orderly arrangement of the blocks and as proof of data integrity till that point. As the hash reference of the last block is actually a hash of data of last block and reference to hash of its last block, that in turn is hash of data and hash of its last block, and so forth. This way all blocks get linked and if a small change is made in the dataset, it will result in breaking the entire link. Thus chain links the blocks in an orderly fashion along with proof of tamper resistance. However, there will be the cases where two block will be generated at the same time. At that moment, there will be two heads of the chain. It is termed as soft fork in blockchain. After the fork, miners / validators have to decide for what chain to extend on? There are different strategies used by various blockchains. Mostly, proof of work based chains select the longest chain with reference to block zero (genesis block) having the maximum computation done for finding the correct hashes.

2.2.3 The Network

Blockchain can be run on a network of computers or a single computer. But, to achieve the goals like data immutability it must run in a distributed manner. Usually, peer to peer network connectivity using distributed application architecture is considered as the best practice for the blockchains. Each peer / node participates to achieve the common goal as per roles assigned to it. The role a node performs, depends on nature of the blockchain environment. In permission-less blockchains every node has equal rights, where as in closed blockchain systems (private and consortium blockchains) few nodes perform more roles than others.

2.2.4 The Rules

Every node on a particular blockchain system obeys the set of rules defined in the code base of that system. A blockchain may and may not have same rules. If there are two blockchains with identical code base, then those chains are either competing chains (soft fork) 2.2.2 or those must be running on exclusively different networks. However, if one alters the code base and is able to participate in the same network as run by the nodes with un-altered code, then such code bases have compatible rules. But there may be the cases where the nodes with modified code will not be able to participate in the system run by nodes with unaltered source code, in this case we say a hard fork has happened. This will lead to two completely different blockchains with each having its own set of rules to verify and validate transactions.

2.3 Consensus Mechanisms

Consensus mechanisms are the backbone of blockchain. These provide a set of rules and procedures to maintain a coherent state of blockchain amongst its participants. Literally consensus means reaching an agreement. Wikipedia describes it as "Consensus decision-making is a group decision-making process in which group members develop, and agree to support a decision in the best interest of the whole." [110] It is a vital requirement of distributed systems to reach a single conclusion. Just like distributed systems, consensus mechanism in blockchain allows a node or a group of nodes to securely update the blockchain (append new blocks to the blockchain) using a predefined set of rules. Blockchain consensus outshine traditional distributed system consensus by allowing distrusting nodes with pseudonymous identities to reach eventual agreement while participant may join or leave the network over time. Earlier for nodes to reach consensus those were supposed to have established identities.

Now that we know consensus definition, one ponders, why those are important in distributed systems? Consensus algorithms provide means where to handle byzantine faults and fail stop fault. Fail stop faults prohibit system to participate in the consensus due to any reason like due to power crash. Such type of fault tolerant systems do not cater for illegitimate responses. Byzantine fault tolerant systems covers a super set of faults. Those do cater for fail stop failures and nodes participating and providing illegitimate responses as well. There are few algorithm those are designed to only handle only fail stop faults. An example of such consensus algorithm is Raft [84]. But in blockchain and related architectures, mostly the consensus algorithms are be byzantine fault tolerant.

2.3.1 Consensus Families

Consensus algorithms can be grouped in different families based on their intrinsic and extrinsic properties. Mostly those are classified on bases of environment those operate an extrinsic property and the way those reach achieve consensus an intrinsic property. Based on the way consensus are achieved, those can be classified as:

- *Proof of X consensus*. Proof of X consensus achieve consensus by allowing nodes to validate transactions and add new block to blockchain with a probability that is proportional to their invested resources in the blockchain system. These resources are different for various blockchains. Proof of work and proof of stake are two major

consensus algorithms from this family and rely on computational resources and stakes in the system, respectively.

- *Vote based Consensus*. In vote based consensus, validating nodes have voting power associated to those. Those nodes cast their vote for the correctness. If the vote received exceed some predefined limit then that transaction is taken as valid. Usually these protocols have a leader to orchestrate the voting process and runs in phases. Two famous algorithms from this family are Practical Byzantine Fault Tolerance and Smart BFT.

- *Hybrid Consensus based on PoX and Vote*. These are hybrid of Proof of X and Vote based consensus. These protocols use proof of x to select members with voting powers and leader and then reach final consensus using some vote based consensus. It should not be confused with other commonly referred hybrid protocols where it refers to a hybrid of Proof of work and proof of stake protocols.

2.3.2 Consensus Evolution and Proposals

Need of consensus arose with invent of distributed systems where rather than a single authority, multiple machines were involved in decision making. In 1982, paper titled "The Byzantine General's Problem" [72] proposed a thought experiment to resolve byzantine behaviour. Paper discussed fictional Byzantine army and proposed a message mechanism for the Byzantine army to reach a reliable decision provided $\frac{2}{3}$ or more of its generals are honest. First efficient and practical implementation for byzantine fault tolerant algorithm came 1999. It was titles "Practical Byzantine Fault Tolerance" and it provided means for programmers to build secure and byzantine resilient distributed networks. Same year (1999), proof of work was proposed [58]. However, it did not find any use beyond email spam handling. Since then, various authors proposed different solutions. All such solutions targetted private distributed network. However, with introduction of bitcoin's Nakamoto consensus (first major use of proof of work) in 2008 [82] for the public networks, the idea of byzantine resilient consensus mechanisms got a new life and academia and industry around the globe proposed numerous proposals and a diversity of algorithms for building consensus mechanisms. These algorithms target various parameters like performance, security, consistency and failure redundancy. All proposed and implemented solutions are based on trade offs.

Those target various settings and parameters. So far, there is no perfect consensus algorithm and perhaps there will never be.

Chapter 3

Selected Parameters

This chapter identifies and describes relevant parameters for evaluation and comparison of consensus algorithms in distributed computing and blockchain. These parameters are grouped under three sections titled security, performance and general. Security parameters define the security guarantees of a blockchain. It discusses adversary tolerance model, consensus finality and ability to handle forks - thus the double spending. Section on performance describes the parameters to judge the performance of a consensus algorithm. It includes parameters like protocols ability to scale both for number of nodes and transactions, operational environment best suited for the algorithm to produce optimal results, block size and puzzle to solve, if any. In section on general parameters, parameters that does not effect the security and performance but are important while selecting a chain are discussed. It discusses about availability of source code for the algorithm, deployment on major systems, incentive mechanisms for joining network and staying honest.

3.1 Security Parameters

Security of a system is context specific. From a very traditional perspective it can be regarded as a combination of Confidentiality, Integrity and Availability [62, 12]. In some other scenario, one can argue that security can be obtained from a combination of identification, authentication and authorization. So context affects the requirements of security of a system. Blockchains inherently provide integrity due to their immutable nature and availability being distributed system with data set replicated across multiple machines. Few may argue that this replication will lead to a weak confidentiality and thus the overall security. But in Blockchain, it is not. It is vouched as to provide transparency. When considering blockchain security, usually following parameters are considered.

3.1.1 Adversary Tolerance Model

Similar to distributed systems, it is the maximum proportion of the underpinning resource of the reference blockchain that adversary can orchestrate without jeopardizing the operation

of the reference blockchain. Number of honest nodes should be at least one more than this fraction to produce true results in presence of dishonest malicious or byzantine nodes. Mostly all the consensus algorithms define a threshold to operate consistently in presence of a certain fraction of byzantine nodes to work correctly. Once this threshold is crossed, results produced may not be true. A higher value is a desirable factor, as higher the threshold, more resilient system is to adversarial / byzantine nodes.

3.1.2 Consensus Finality

Consensus Finality is a property that mandates that once a valid block has been appended to the blockchain by any valid node in the network it can not be reverted. It heavily relies on consensus design and is a function of threshold byzantine nodes a protocol can handle for production of true results and minimum number of transactions required to be sure that a transaction has been finalized in the blockchain. If we talk of bitcoin, it can tolerate upto 50% byzantine nodes and provides confirmation after 6 blocks. But if the byzantine nodes are more than 50% then even those 6 blocks can be reverted, thus destroying finalized transactions as well [49]. If we compare consensus finality to standard distributed computing, for an atomic broadcast it is similar to total order and agreement properties [35]. Consensus Finality impacts latency of the system and determines whether forks happen in a blockchain system or not? It is crucial as it provides certainty whether a asset legally belongs to an entity or not? It is especially important in financial sector where it maps directly to an economic loss. In Proof of Stake term Economic Finality is also used that tells how much economic loss will happen if some blocks get reverted.

There are two types of consensus finlalities in blockchain namely Absolute Finality and Probabilistic Finality. **Absolute Finality** In absolute finality once a transaction is added to the chain, it is immediately finalized. It will stay part of the chain forever. **Probabilistic Finality** In probabilistic finality once a transaction is added to the chain, it is not immediately finalized. Instead we wait for it to get deeper in the chain to be finalized. The more deeper it get, there will be lesser chances that transaction get reverted. For Bitcoin it currently 6 blocks[42].

3.1.3 Forks Handling

Owing to decentralized data structure and latency in the network, all nodes may not have consistent copy of the chain. Thus based on how early a node receives the newly created block, it will have a different perspective of the chain at that time. Now lets imagine for a moment that two nodes, node nA and node nB generated two blocks bA and bB simultaneously at the same time. After the block has been produced, it will be propagated immediately to other neighboring nodes in the network who will incorporate int their copy of the chain and will propagate the same block to other neighboring nodes. As the two blocks were produced at the same time, few nodes will have block bA as last block and few will have block bB. Thus we can say a soft fork has happened in the network as there are two competing chains with different last block. Every consensus algorithm should provide some rules to

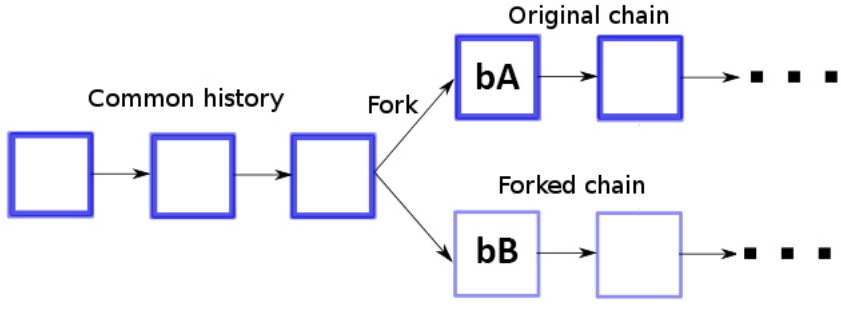


Figure 3.1: Fork Creation - Competing Chains

handle such soft forks.

3.1.4 Puzzle Design and Design Goals

Puzzle Design Puzzle design defines mechanism of leader election. The elected leader adds transactions to the block and then appends that block to the chain. In Proof of Work based algorithms this selection is done using some kind of one way hash function and a challenge to make the puzzle hard. Usually the one way function used is a non deterministic polynomial time (NP) hard problem such as SHA-256. It is very easy to verify the output of the NP hard function but not possible (at least not feasible as it requires brute forcing the entire space) in polynomial time to go from output to input. In Proof of Stake it is based on stakes in the system and in Byzantine Fault Tolerant Systems on some pre defined parameters. **Design Goal** What puzzle is picked up for use in the algorithm is heavily influenced by the

environment system is to be deployed in and on some specific goals like scalability, hardness, usability , ASIC resistance, environment friendliness, some contribution to mathematics or may be some other.

3.1.5 ASIC Resisability

Application Specific Integrated Circuit (ASIC) is a silicon chip specifically designed to perform a single task in an efficient way. ASIC Resisability is a desirable property for blockchain as it mandates the core philosophy of blockchain decentralization. It will not only provide all miners a fair playground but will hinder operation of large-scale ASIC farms. Thus it can effectively impede formation of mining pools. Initially ASIC Resistant algorithms used memory hard puzzles such as Scrypt where a large amount of memory is required to solve the puzzle. However, as memory is becoming cheaper and cheaper and it is also possible to manufacture nano scale circuits, ASICs have become available for memory hard puzzles. Another approach suggested using chained hashing schemes such as X11 as designing of multiple hashing functions on ASICs is not feasible. But ASIC miners have become commercially available for such schemes as well. Another suggested approach to achieve ASIC Resisability is using self-mutating puzzles. These puzzles change the underlying hard puzzle intelligently or randomly as function of time[14].

3.1.6 Anonymity and Privacy

Anonymity Oxford Advanced Learners Dictionary defines anonymity as ”the state of remaining unknown to most other people.” And it means ”Not named or identified” according to Merriam-Webster. Most blockchains offer pseudo anonymity and achieve this by privately assigning unique pseudo randomly generated keys to a user. A user can generate an many of these private keys for himself as desired. These key are stored in software or hardware wallets. Thus original identity of the user is masked behind multiple keys. However, due to public nature of permission-less blockchains these identities only provide limited anonymity or more precisely linkable identity. Many solutions have been proposed and implemented like Coin Mixing [91] to enhance the anonymity but that is beyond the scope. **Privacy**Privacy is strongly related to confidentiality and is a fundamental right of individuals. It defines a person’s right to limit the degree to which it is willing to interact and share information with the specified system. However, not all Blockchain especially permission-less provide

privacy. As these systems are transparent and all their transaction are announced publicly to the system [9] thus allowing attackers to construct back individual identities[63]. In this theses, we shall look for what all anonymity and privacy measures are supported by Blockchain consensus.

3.1.7 CIA and Blockchain

Confidentiality, Integrity and Availability has not been included due to reasons specified in below paragraphs. **Confidentiality** Confidentiality defines that unauthorized disclosure of data should not take place. However, in case of public blockchains, those believe in transparency where all data is available to all the full nodes in the network [89]. Identities are hidden using some pseudonyms those can be easily traced back to original identities. Also if symmetric or asymmetric cryptography is used for encrypting transactions, those have to be decrypted at all nodes for smart contracts to run [114]. **Integrity** Integrity is a key aspect of blockchain and defines inability of unauthorized systems or nodes to alter data in some way . Immutability of blockchain transactions enhances with more blocks confirmed to blockchain. Thus, it can be concluded that blockchains provide integrity property. Same is true for the smart contracts. **Availability** Availability is readiness of a service for correctness when required [12]. Data in blockchains are massively replicated, at least in case of public blockchains. Thus there are thousand copies of same data available around the globe all the times[85]. Thus it can be considered that there will be very less chances of non availability unless a specific node or a collection of nodes comes under some kind of attack.

3.2 Performance Parameters

3.2.1 Block Size

Block size defines whether a blockchain mechanism defines the size of a block or its variable.However, mostly all blockchain structures specify it. Provided the block interval is fixed, block size impact the following:-

1. **Throughput**. Throughput is directly proportional to Block size as number of transactions are dependent on it.

2. **Capacity**. Larger the block size, the capacity to transfer data will be increased as well.

3. **Scalability**. Increase in block size will increase in number of transaction thus system making more scalable to today's needs.

4. **Security and Forks**. Small block size will have very less number of transactions and will be generated quickly, thus may lead to multiple soft forks. This in turn will lead to more orphaned blocks that will impact transaction's security.

5. **Propagation**. Larger the block size, slower the propagation.

6. **Congestion**. Larger block size will increase in congestion rates.

3.2.2 Puzzle To Solve and Block Minting Mechanism

In public (permission-less) blockchain network no third trusted party exists. That missing third party trust is achieved by the puzzle that all miners solve to find the solution. Thus the puzzle selected or designed for the nodes to reach consensus must guarantee freshness at the time of execution. That means it solution should be unpredictable and proof is not reusable. It should also meet properties mentioned in 3.1.5 and **??** Block minting mechanism tells how new blocks will be created for the blockchain. Usually it is referred as "mining" [14] because in bitcoin Proof of Work[82] algorithm, the most famous crypto currency termed creator of new blocks as miners, and "forging or minting" [57] due to its reference in the most famous blockchain platform Ethereum [111]. It is as crucial as puzzle design itself as transaction verification and validity [21] is done through this process. Block minters are referred with different names in various algorithms.

3.2.3 Scalability

Scalability of blockchain can be viewed under two sub parameters i.e network scalability and transactions scalability. **Network Scalability** Network scalability defines how nodes join or leave the network and its impact on networks performance. Network portion is very scalable for the permission-less blockchains. New user can join the network without impacting its performance. **Transcation Scalability** Transaction scalability defines how many transaction can be performed in a minute. Mostly transactions scalablity for all major blockchain systems is limited [106] due to factors discussed in 3.2.1 and [96]:

1. Time taken to reach consensus.

2. Time taken to put transaction in block.

3.2.4 Operational Environment

Operational Environment defines the domain where a particular blockchain can be deployed. It can be Private or public. Private domain blockchains are either permissioned or consortium. In permissioned environment only a single entity has the right to right transactions to the chain where in consortium same right is distributed amongst a group of nodes. In public or permissionless environment, any one who can prove the specific challenge can add transactions to the chain.

3.2.5 Imposed Restrictions

Certain blockchains impose few restrictions on itself to achieve some specific goals like security, enhanced performance or any other deemed necessary for the efficient functioning. These parameters are very deliberately weighed out after test and trials to be part of the chain like block size in Bitcoin is fixed to 1 MB or upper limit for gas in Ethereum is 6.7 million per block [40].

3.2.6 Communication Model

Communication model is an important aspect of blockchain security and directly impacts consensus' trust. As in most distributed system, two most widely communication models used in blockchain are synchronous and asynchronous. When synchronous communication model is used, sender has to wait for the recipient for acknowledgement of the request. Thus we can say trust in explicit, that is trust is achieved once both parties agree on the common results. Contrary to this, in asynchronous model trust is implicit. In asynchronous model, sender will continue communication without waiting for the recipient acknowledgement. Thus once sender has performed some transaction, recipient will accept it as honest and true transaction.

3.3 Generic Parameters

3.3.1 Handling of Digital Assets

It defines whether the referenced blockchain has some digital currency linked to it like bitcoin or some real word currency or something else that gives it a value. It is not necessary that all blockchain networks are linked to some type of currency or some other asset. It may

just be a platform for decentralized applications or even it can offer both a digital currency and a platform like Ethereum.

3.3.2 Handling of Smart Contracts

Smart Contract for understanding purposes may be referred as stored procedure in a traditional data base [114]. But in reality smart contracts are much more powerful. Those digitally facilitate in verification, enforcement of a negotiation or a contract's performance. Transactions are only performed once required parameters are met. [109]. Not all blockchain support smart contracts. Only a few like Ethereum and Hyperledger support it. Few have limited scripting support like Bitcoin. For distributed applications to harness the power of blockchain technolgy's distributed architecture, blockchain with smart contract support is needed.

3.3.3 Incentive Mechanism

In peer to peer distributed applications, nodes self organize and cooperate amongst themselves to effectively run the blockchain network. These nodes have to validate transaction before adding those to blockchain besides propagating those to network and storage space for the chain data. Block creation is itself a resource intensive task. It consumes a lot of electricity or monetary assets besides bandwidth requirements, hardware maintenance cost and others. To meet these requirements of the nodes and make the block creation and validation profitable all blockchains define some incentive mechanism for the nodes.[51]

3.3.4 Deployment

This describes the easiness for deployment of a blockchain. This will depend on which all major operating systems (Windows, MacOS, Linux) it supports. What all external libraries are to be installed. How easy it is to configure the system.

3.3.5 Open Source and Related Projects

This tells whether the referenced blockchain is open source or close source. If it is open source, is its code available on famous project hosting sites like github and sourceforg. Also what projects exist on github related to the referenced blockchain.

3.3.6 Documentation and Public Forums

Documentation and Public Forums describe the documentation available to the developer including online resources. This includes platform design documents, documents related to implementation and its features, working examples and tutorials[52].

3.3.7 Accessibility

Blockchain provides real time accessibility that is data store on the chain is available to nodes through multiple nodes along with a local copy but by accessibility here means ways through which referenced blockchain can be accessed. This describes all the methods those are available to interact with the blockchain. It may include some specific libraries, REST API, remote procedure call or some graphical user interface. This parameter determines how easy it is to use a specific blockchain platform.

3.3.8 Blockchain Type

Blockchain is classified into three categories based on accessibility 3.3.7 and it limits how membership to a chain is controlled. These three types are:

1. Public Blockchain : Blockchain those can be accessed and updated publicly.
2. Private Blockchain : Blockchain those can be accessed by a single entity for adding block to the chain.
3. Consortium Blockchain : Blockchain those can accessed and managed by a group of entities.

This classification determines what blockchain is suited for what type of business.

Chapter 4

Consensus Based on PoX Schemes

4.1 Proof of Work - Nakamoto Scheme AKA PoW

Satoshi Nakamoto presented first consensus protocol to work on public networks in open settings based on *proof of work*. The idea of proof of work was originally published in 1993 by Cynthia Dwork and Moni Naor and Markus Jakob and Ari Juels coined the term *Proof of Work* in year 1999. It proved to be the biggest idea behind Nakamoto's Bitcoin paper [82] published in 2008 and formed bases for the distributed trustless consensus. Nakamoto consensus was designed to ensure security in a trustless environment without need of intermediaries. Protocol is open source, has been heavily reviewed, updated and till to date the most widely used protocol for public consensus [60]. Nakamoto consensus has evolved over time and has been maintained to adapt to environmental changes. Future of the Bitcoin (Nakamoto consensus) is governed by the nodes with large hashing powers. Updates to the protocol are proposed through Bitcoin Improvement Proposals, most commonly referred as *BIPs* [73]. Once a BIP is proposed, the miners cast their vote for the acceptance or rejection in the coinbase transaction. Based on majority vote, either system moves to new codebase or rejects the proposed BIP. Mostly the new codebase is backward compatible (termed *Soft Fork*) and sometimes it is not (and is termed a *Hard Fork*) The nodes with more hashing power govern the. Nakamoto consensus formed the bases for crypto-currency called 'Bitcoin' and is the mostly widely accepted crypto-currency today. However, recently it has come under heavy critique for its limitations especially throughput, latency and transaction finality. These limitations are attributed to the underlying consensus protocol. Bitcoin achieves almost 7 transaction / second and it requires approximately 6 blocks to be sure that transaction will not be purged due to forks. Every block has a latency of about 10 minutes, thus to be safe that transaction will not be reverted, user has to wait for an hour, almost. These factors make it a non viable option for the commercial sector. However, few modifications have been proposed by various authors to over come the limitations .

4.1.1 Working

Concept In Nakamoto consensus every network participant has one or more private identities associated with it. Various nodes create transactions amongst each other. Those transactions are added to a central pool of transactions. Every miner node collects transactions from the pool and tries to add those to a block after verification. Probability that a node will be able to add the block to the blockchain is proportional to the computational power it has invested in the network to the total computational power of all the nodes currently in the network. The mining node win the lottery to add block to the blockchain by employing a hash function Double SHA-256 to calculate the hash of the block header a random number termed nonce. If the resultant hash is below a certain predefined threshold, miner wins and gets the reward for finding the valid block and adds that block to the blockchain. All other nodes, implicitly accept that block as a valid block and start computing hash for the next block. This hash finding process is termed as 'Proof of work'. **Working** Nakamoto consensus or Mining plays multiple roles in Bitcoin. In fact, it is the main idea that underpins the Bitcoin creation. It allows nodes on an untrusted network to agree on a universal truth of ownership, yet without trusting everyone. In Bitcoin network, every node can have multiple private and public identities. Number of identities does not matter as the ability to mine is proportional to total computational power invested in the Bitcoin network compared to all the computational power invested in the system by all other nodes active on the network. Based on the mining capability, nodes on the network can be categorized into two shades:

1. *Full Nodes*. These nodes have a complete copy of the public ledger (data set for the chain). Thus these nodes can themselves verify new transactions by reading the ledger. These nodes can participate in mining process.

2. *Light Nodes*. These are client nodes which do not have full copy of the ledger. These nodes get help of neighbouring nodes for verification of any transaction those are involved in and can not participate in mining process.

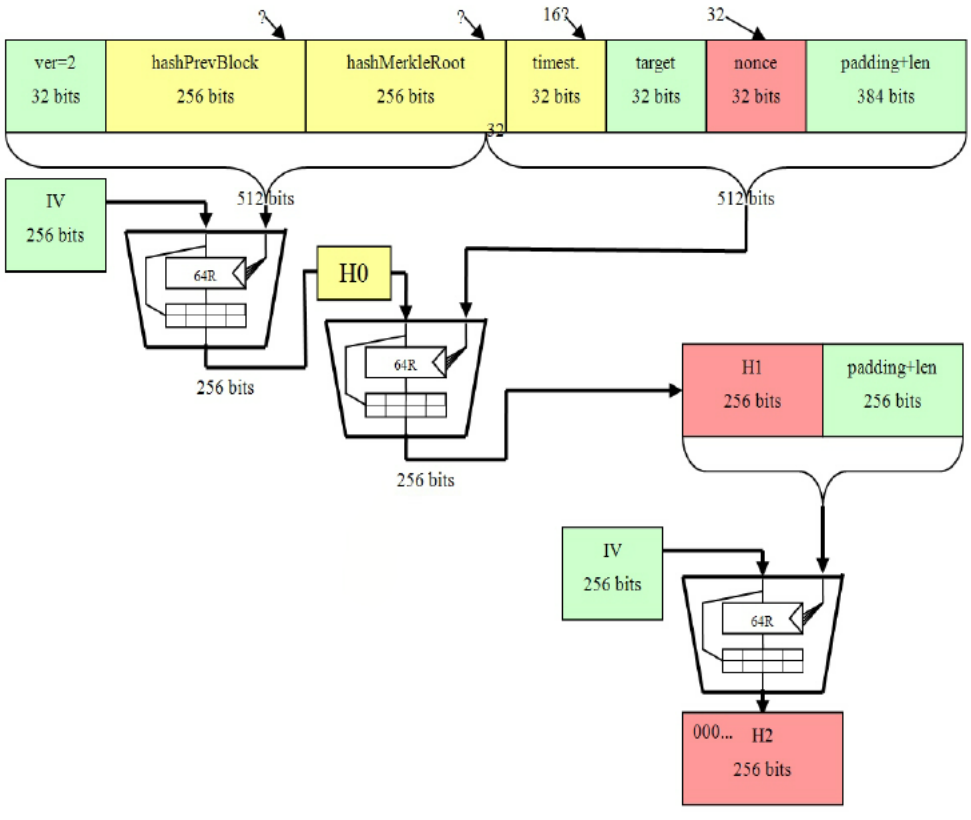
Only full nodes participate in mining and hence the consensus process. All the full nodes on the network independently execute four processes based on simple rules to achieve consensus on a unique global state of the publicly available ledger:

1. *Independent Verification of Transactions*. Every full node verifies each transaction independently based on a comprehensive list of criteria like transaction syntax, inputs and outputs and transaction size to name a few (these conditions change over time and can be found in functions *AcceptToMemoryPool* in Bitcoin Core [10]. If the transaction is found valid it is then forwarded to the neighbouring nodes. Each of these nodes will verify the transaction again and if found correct it will be forwarded to the neighbouring nodes and process will continue unless a valid transaction has propagated the entire network or an invalid transaction has been discarded. Valid transactions are added to *memory pool* or sometimes called *transaction pool*. These transactions have been validated but yet to be confirmed.

2. *Independent Aggregation of Transactions*. Mining nodes try to aggregate transactions from the mempool to a *candidate block*. Miners try to find the solution for proof of work to validate the block. Proof of work is calculated by repeatedly finding SHA256 hash of the block header and random number termed none, unless the resultant hash is less than a certain threshold as can be seen in fig **??**.If a miner successful in solving the puzzle, he will propagate the block to the neighbouring peers.

3. *Independent Validation of the Block*. Once a miner has found the block and propagated it to the neighbouring peers, then these peers will validate the block against a set of criteria and include things like block structure, syntax validity, hash of the block, timestamp. Complete list of these criteria can be seen in *CheckBlockHeader* and *CheckBlock* function of BitCoin Core. This independent verification disallows miners from cheating. Once a block has been validated, it will be propagated further to the peers unless it ripples out to entire network.

4. *Independent Chain Selection* Once a full node receives a new valid block, it will add it to its existing blockchain. If two conflicting valid blocks are received at that refer to same last block then that block will be kept as sibling to the other block that referred to same last block. Node will wait for another block. It will extend one of the forks happened previously. Thus this node will implicitly decide to keep the chain with most cumulative computation done in solving proof of work puzzle [14].



Multiple Hashing scheme for Nakamoto's Proof of Work [10]

4.1.2 Security

Adversary Tolerance Nakamoto Consensus can tolerate up to 25% adversarial computational power. However, most of the literature mentions it to be 50% or $2f + 1$ nodes where f are the byzantine nodes [75]. That is true from theorist point of view where more than half of the computational power is required to create a competing blockchain with most computational work on it. However, in reality with 25% adversary existing on the network, the transaction finality is shaken and after 11 blocks it achieves only weak finality i.e transacting node has to wait almost 2 hours after the transaction has been added [102] to be mildly sure the transaction will not be reverted. It requires addition of 26 blocks to the blockchain for a strong finality and to be almost 100% sure about transaction confirmation. **Consensus Finality** Nakamoto Consensus provides probabilistic guarantee for settlement finality [105]. This means transactions those have been finalised and have become part of the chain can be reverted. It can happen due to many reason like:

- Because of a vulnerability in software that adversary can exploit.
- Different client applications may accept different chains as longest chains. Similar

Algorithm 1: Nakamoto Consensus Pseudo Algorithm

```
Input: A set of valid transactions
Output: A valid block
/* Join the Network                                                    */
Join the Blockchain Network as a Full Node (Mining Node)
Function Main():
    boolean blockReceived=false; boolean blockFound=false
    while !blockFound do
        validate new transactions
        if Transaction is valid then
            gossip to peers
            add to mempool
        else
            destroy transaction
        end
        if !blockReceived then
            run GenerateBlock()
        end
        validate the newly received or generated block
        if Block is valid then
            gossip to peers
            select the longest chain
            append block to this chain
        else
            destroy the block
        end
    end
    /* Participate in block generation (mining process)                */
    Start BlockGeneration()
End Function
Function GenerateBlock():
    collect transactions from mempool
    pack transactions in block
    generate block header
    while block is not found do
        solve proof of work hash puzzle by concatenating nonce and blockheader
        if block is found then
            set blockFound = true
        end
    end
    return new block
End Function
```

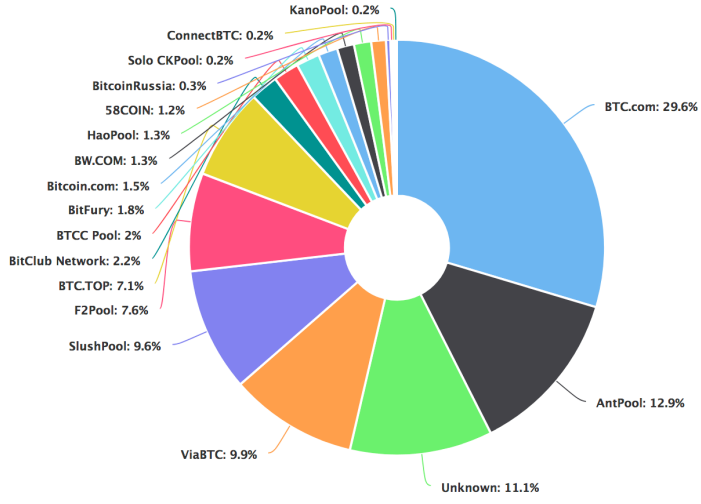
incident took place in 2013 where one version of Bitcoin accepted one chain and other version of the software rejected the same chain though it was the blockchain with most computation [104].

- Malformed block generation by the miners. In 2015, roughly 6 blocks of a mining pool named F2P got reverted due to this [5].

Based on these facts, there is a strong argument that Nakamoto Consensus can never achieve consensus in reality. That may be true from theorist perspective but in reality it sufficiently reaches close to finality once 6 blocks have confirmed i.e the block with the transaction and 5 blocks after that. **Forks Handling** Owing to distributed nature of the Bitcoin network, forks may arise due to network latency or other issues like errors in codebase or induced deliberately by some attacker. When a fork happens, each node independently tries to resolve that by extending the longest chain with the most proof of work i.e the longest chain with the most cumulative work. Nodes can calculate the total computational power gone into to creation of a blockchain by adding computational power for each block of the referenced blockchain. As long as the majority of the nodes select the chain with most cumulative proof of work, the network will converge to a consistent state after few blocks. **Double Spending** Double spending is possible in Bitcoin due to latency in block confirmation, soft and hard forks or flaws in implementation in underlying cryptographic primitives. However, the likelihood diminishes as more and more blocks get added to the main blockchain and hard forks stay compliant with older versions [46]. **ASIC Resistibility** Nakamoto Consensus is a non ASIC-resistant proof of work scheme. Miners can use specialised ASIC chips to compute the double SHA-256 function. This lead to the formation of ASIC farms that challenges the idea of decentralization. Specific to Bitcoin that makes use of Nakamoto Consensus, it is controlled by only few mining pools as shown in fig 4.1. It is pertinent to note that more than 50% computational power is controlled by only three mining pools [61].

4.1.3 Performance

Block Size Bitcoin specifies the block size of 1 MB produced about every 10 minutes. Varying block size impacts the throughput and security of the system. Research by [112] have shown that by keeping the block size of 1 MB and reducing the block production rate to 1 block per minute does not much impact the existing security and yet is able to achieve 60 transactions per second. That can be a considerable improvement to existing system. **Throughput** Nakamoto Consensus gave a new life to distributed computing for achieving trust among the non trusted participants in absence of federated identities and trusted third



(a) More than 50% Computational Power is controlled by Three Mining Pools

Figure 4.1: Bitcoin Mining Pools

party. However, it failed utterly to scale out as compared to traditional system. Its purest implementation, Bitcoin achieves approximately 3.3 to 7 transaction in a second [33]. Thus making it non usable in high traffic system without modifications. **Scaling** Nakamoto Consensus in its purest form as used in Bitcoin does not scale well. Usually for a technology to scale out, more hardware is added. However, that is not true for Bitcoin. Here the puzzle design is adaptive. As the more and more hardware is added, the complexity to solve the cryptographic pre image search puzzle increases, and time to mine the block remains almost same, that is approximately 10 minutes. Also the size of the block is fixed to 1 MB to achieve high security. These factors inhibits Nakamoto Consensus to scale out 4.2 with more computational power added to the network [14, 66]. **Operational Environment** Nakamoto Consensus has been designed to provide consistent results in permissionless environments where any one can join or leave the blockchain network at will and have no pre-established identities. **Communication Model** Bitcoin uses gossip or epidemic protocol [108] for peer to peer communication. When a node generates a message, it propagates a message m to i active neighbouring peers. Each of i neighbouring peers on receiving the m for the first time, independently and randomly select a number of his neighbours and forwards the m to those and does so until receive a confirmation. This propagation mechanism terminates once no user receives the message m for the first time.

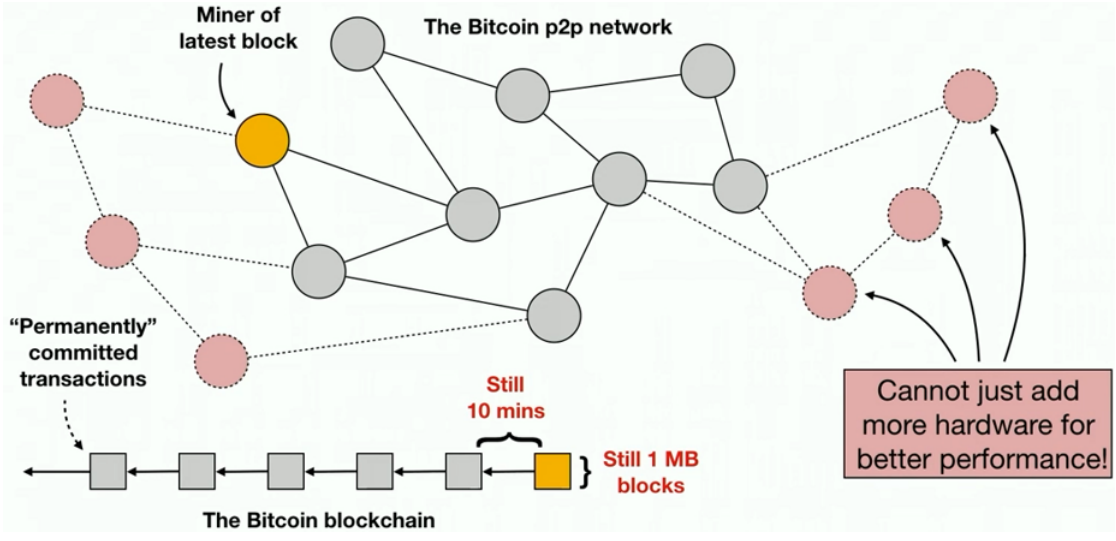


Figure 4.2: Scaling impact of More Computational Power — Hardware

4.1.4 Generic

Incentive Mechanism Miners are incentivised through issuance of new bitcoin generated at the time of block creation and transaction fees. A miner who generates the new block will get all the transaction fees offered by transaction making parties plus the a fixed amount of coins minted as per protocol at a diminishing rate. When a new block is created, it contains new coins generated from no where. These new coins are produced at a fixed rate for a fixed interval of 210,000 blocks. As a block is produced usually after 10 minutes, that makes it almost 4 years. After that interval, new minted coins decrease by 50%. In year 2008, new block minted 50 bitcoins. In 2012, 25 coins were minted by each block and in 2020 it will become 6.25 bitcoins per block. Finally, it will diminish to zero once 6.93 million blocks have been produced [10]. At this stage, miners will collect all rewards from the transactions fee.

4.2 Proof of Work - Bitcoin-NG

Bitcoin-NG [39] is a slightly modified version of Bitcoin[82] based on Nakamoto's Proof of work 4.1. Thus it inherits all the properties of security and performance from Nakamoto Scheme less scalability as it has been designed with an aim to add scalability to its parent protocol i.e Nakamoto Consensus. And how Bitcoin-NG achieves scalability? It makes two modifications. First it decouples Bitcoin's mining process into two planes i.e leader election (production of key blocks) and transaction serialization (creation of micro blocks).

Second time is divided into epochs and for each epic a single leader is elected through traditional proof of work. The elected leader has the responsibility to unilaterally serialize all transactions (produce micro blocks) until a new leader is elected. Or in other words, at start of an epoch the miner or leader of Nakamoto consensus is selected through solving hard proof of work puzzle. He then adds several transactions (micro blocks) to the chain for the entire epoch by cryptographically signing those without doing any extra proof of work[103]. Once a new leader / miner has elected the puzzle solving, he / she decides on what previously added blocks chain has to be built on [76]. Source code for Bitcoin-NG is not available. It is pertinent to mention that it is backward compatible with Bitcoin i.e it can operate with wallets and clients used by Bitcoin.

4.2.1 Security

Bitcoin-NG offers security very similar to its master Bitcoin. It gets strength from proof of work hard puzzle where it requires solving a pre-image search function based on SHA-256 with specific parameters like number of zeros at start of output hash and incentive compatibility by motivating network participant to honestly follow the rules[39]. It guards against a maximum of 1/4 of the total Byzantine computational power i.e if there are $\mathcal{N}$ nodes in the network and at any time t a subset $\mathcal{B}(t)$ behaves arbitrarily and are controlled by a single adversary then the mining power m of the honest nodes must be more than 3/4 of total mining power of $\mathcal{N}$ nodes for system to produce true results:

$$\forall t : \sum_{b \in \mathcal{B}} m(b) < \frac{1}{4} \sum_{n \in \mathcal{N}} m\,(n)$$

Selfish Mining Bitcoin-NG is susceptible to all attacks same as Bitcoin [45, 11]. It is pertinent to mention that Bitcoin-NG is very vulnerable to selfish mining. If the attacker controls more than $\frac{1}{4}$ of the total network, he can launch self mining attack [93]. **Double Spending** Double spending is possible in Bitcoin-NG due to forks and signing of micro blocks from a single validator. If a validator adds few genuine blocks at start of epoch and later adds blocks with malicious transactions. These malicious transactions will only be detected later a key block is mined by a new miner. However, by that time transactions have already been added to the blockchain by the compromised miner. The mitigation is to introduce the fork. **Forks Resolution** Forks can happen at key block and micro block level. However, at micro block

Algorithm 2: Bitcoin-NG Pseudo Algorithm

Input: A set of valid transactions and hash of previous block
Output: A Key Block and Several Micro Blocks

```
Join the Blockchain Network as a Full Node (Mining Node)
Function Main():
    P = Hash of the last block
    Bk = Key Block containing P, current GMT,coin base reward, target value and
     nonce
    Bm = Micro Block containing ledger entries
    H = Hash function
    D = Difficulty level
    retrieve P
    while Bk is not found do
        Construct Bk
        /* will be true once key block has been found                    */
        if H(P,BK) < D then
            Block is found
            gossip Bk to peers
            /* create micro blocks                                       */
            run GenerateBlock()
        end
        update nonce
    end
    if Bk is received from other node then
        validate Bk
        if Block is valid then
            gossip to peers
            select the longest chain
            append key block to this chain
            append micro blocks from leader node to the chain after validation
        else
            destroy the block
        end
    end
End Function
Function GenerateBlock():
    while Bk is not found by other node do
        collect transactions from mempool
        pack transactions in block
        generate block header
        sign the block
        append the block to the chain
    end
End Function
```

level, as micro blocks become more frequent, possibility of shorts forks becomes more pronounced. To handle both the issues, algorithm has defined few mechanisms. For Key Blocks it resolves forks by accepting the chain which aggregates the most work done for all the key blocks in the network. For micro blocks as those do not involve mining and can quickly and cheaply be produced by leader, resolution is done through leader incentive mechanism. Algorithm has introduced a dedicated ledger entry to invalidate fraudulent leader's revenue. As new miner of the key block has to decide what what previous micro blocks to build the chain on, he may prune the chain for fraudulent behavior adding only header of the first block of the pruned chain in the new chain. This entry is referred as *poison transaction*. The minor who will mark such transaction will get extra 5% from the fraudulent leader's account as an incentive.

4.2.2 Performance

Performance is effected by network conditions and processing power of the nodes. That is, it varies with size of the network and number of micro blocks as it can be seen in Figure 4.3. Bitcoin-NG inherits Bitcoin, thus its propagation model as well [39]. All the transaction are to be broadcast to all the nodes in the network. Thus the network latency is increased with more number of nodes added to the network and more micro blocks being produced. With a smaller network size for example 100 nodes and 4 blocks it has a latency of 50 seconds. However, as the network size grows larger its latency increases and performance worsens. With 1600 nodes and 4 blocks it latency increases to 456 seconds. Also the bandwidth requirements of the nodes rise linearly to the throughput [76]. Again the reason is propagation model. Bitcoin-NG has to broadcast all blocks to all nodes present in the network to reach next consensus. Thus as more blocks get added and more nodes join the network, to bandwidth requirement of nodes will keep increasing.

4.2.3 Generic

Incentive Mechanism As described earlier in this section, Bitcoin-NG mining involves production of two types of blocks namely *key blocks* and *micro blocks*. Key blocks are found the similar way as miners find blocks in Nakamoto Consensus [39, 99] i.e by solving a one way collision resistant cryptographic hash puzzle of the block header with a set threshold[39, 62]. A miner who finds the key blocks serialize the transactions into micro blocks by signing

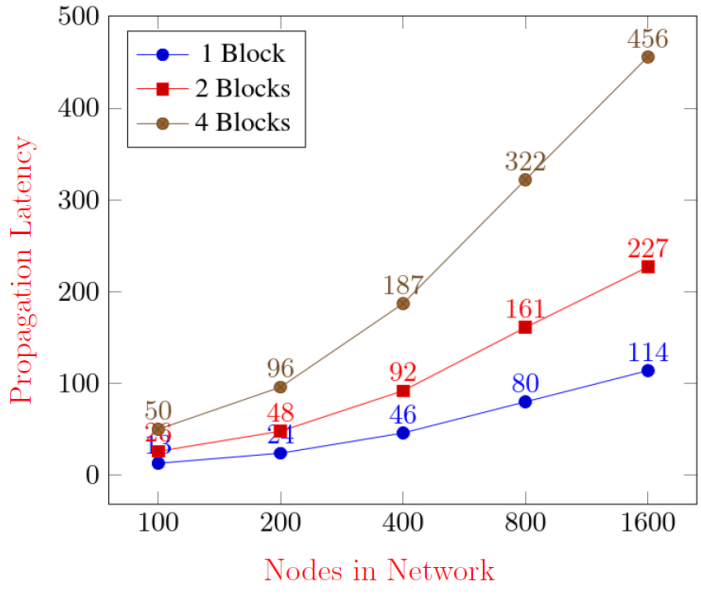
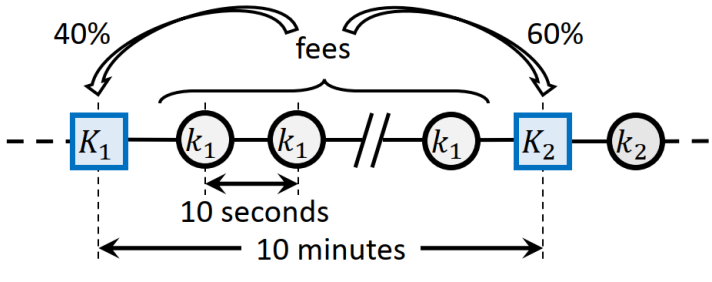


Figure 4.3: Impact of varying number of blocks on Bitcoin-NG's Latency

those with his / her private key without doing any work unless a new key block is found by another or same miner. New miner of the key block (indicated as K_1 and K_2 in square blocks in fig 4.4) decides what micro blocks to extend the chain on. The miner of the key block who serializes the micro blocks get a reward of 40% and the miner of the key block who decides what micro blocks to extend the chain on gets 60% reward 4.4. Key block is found at a fixed interval of 10 minutes. Other than this reward there is fees as well for adding each transaction entry to the ledger. That fee solely belongs to the miner serializing the transactions.



(a) Square Represents Key blocks and Circle represents Micro blocks

Figure 4.4: Bitcoin-NG Incentive for Miners

Coinbase Transaction Similar to Bitcoin coinbase transaction mints new coins and are deposited to miners as described in last paragraph about Incentive Mechanism sharing. Coins created in coinbase can only be spent aftre those have a maturity period of 100 blocks.

4.2.4 Suggested Improvement

Validity from the Start Bitcoin-NG checks for the validity of the transactions once those have been appended to the chain. However, at this point the only option to prune the malicious transactions is to introduce fork in the system. Thus to avoid forking, the system need to validate transactions from the outset. All transactions to be verified before those are appended to the blockchain. **Implementation** Aeternity Blockchain [1] consumes the idea of Bitcoin-NG. **Documentation** Though it got a lot of attention, it has not been well documented. Other than its white paper and material found on aeternity blockchain [7], there is not much present on the web.

4.3 Ouroboros Genesis

Ouroboros is the first provable secure proof of stake protocol (a virtual mining technique where miners can mine blocks proportional to their assets invested in the system) designed to achieve at least security characteristics of Nakamoto Consensus with minimal resource consumption. With it first version released in 2016, protocol is ever evolving to meet challenges of the real world. Its latest version Ouroboros Genesis was released in 2018. It is being actively used in Cardano blockchain and has been extensively peer reviewed. It over comes the limitations of virtual mining techniques like cost-less simulation in proof of stake. Protocol provides a Dynamic Availability environment for nodes to operate. Dynamic Availability setting is a suggested environment where parties can join and leave the network at will, nodes may leave synchronization and network connection and protocol does not have to have a prior knowledge of participation levels. Ouroboros protocol allows nodes to participate in block generation even when those are offline by delegating task to other nodes, allows nodes to go offline when those are not making transactions. This happens because of novel chain selection rule of Ouroboros protocol. The chain selection rules allows participating nodes to bootstrap all information from the genesis block without requirement of any intermediary information.

4.3.1 Working

Concept When the system starts execution, data of all the stakes and their owners are stored in genesis block. System then draws a lottery using a random seed derived from the chain itself using a verifiable random function and follow the Satoshi procedure [15] to determine

the block leader termed as slot leader and the slot in which specific leader will mine the block. **Working** Ouroboros protocol is an evolving protocol. Its latest update is termed as Ouroboros Genesis, because with latest revision any participant can bootstrap from the genesis block without requiring any intermediary data points or some external data. This is contrary to older versions and other existing proof of stake based protocols like PeerCoin [64], NxtCoin[83] and many others. Those protocols require checkpoints or trusted third parties to get a valid trusted chain. Protocol divides the time in slots. Time for the genesis block is T_0 and the block after the genesis block is T_1 and continues to grow. Multiple slots are combined together to form an epoch. Every epoch has a virtual genesis block. Genesis block has fixed stakes and user identities. For the virtual genesis blocks this stakes and user identities can be build by traversing the chain backward. During an epoch network participants can transfer their shares. However, their probability to win the election as a leader depends on their stakes they had at end of last of epoch i.e their stakes in current epoch's genesis block.

At start of the current epoch Ep_0 , block B_0 i.e genesis block holds distribution of all stakes and identities of all the stake holders. This root is taken as trusted by the system. Next every network participant determines whether it is a slot leader or not? This is determined by evaluating a verifiable random function locally using participants public key and epoch randomness obtained passing first $\frac{2}{3}$ verifiable random function proofs P_π from the last epoch Ep_{-1} to a pre-image resistant hash function . If the output (depends on current stakes in the system) of the verifiable random function is below certain threshold, it proves that the particular node is eligible to be slot leader for the particular slot. Now, the eligible node will compute a proof of the output returned by the verifiable random function P_π to certify itself as the leader for the determined slot.

4.3.2 Security

Sybil Proof Ouroboros is designed for the permissionless environment thus does not require any pre-established identities. Nodes on joining network can create any number of identities dynamically. However, more identities does not increase chances for block creation. The probability that an identity (node) will get selected as leader of a slot for block creation depends on the stakes associated with that identity. **Adversary Tolerance** Ouroboros

Algorithm 3: Ouroboros Family Pseudo Algorithm

Input: A set of valid transactions and hash of previous block
Output: A Key Block and Several Micro Blocks

```
Join the Blockchain Network as a Full Node (Mining Node)
Deposit stakes to the chain pool
Function Main():
    while participating for minting do
        if Block is received from other node then
            validate block
            if Block is validated then
                select the longest chain
                append block to this chain
                relay block to others committee members
            else
                destroy the block
            end
        else
            if new BlockCycle has not started then
                participate in ElectCommittee()
                fetch sequence number for my turn to mint blocks
            end
        end
        if my turn to mint blocks then
            collect transactions
            fetch last block hash
            mint block
            append block to longest chain
            relay block to others committee members
        end
    end
End Function
Function ElectComittee():
    while Block is not found by other node do
        fetch stakes of members from the virtual genesis block
        run Multiparty computation protocol
        return block generation sequence
    end
End Function
```

protocol offers high security against adversary as compared to other protocols. It tolerates malicious nodes upto 49% stakes present in the system. Unless the honest nodes have at least 51% stakes in the system, the network will achieve liveness and safety irrespective of network transaction delays and temporarily synchronization issues. **Forks** Fork resolution is very unique in Ouroboros protocol. Like Bitcoin contrary to other proof of stake protocols

network participants can boot strap from the genesis block to the latest (tail) block without requiring any intermediary. In Bitcoin forks can be resolved by following the longest chain i.e the chain with maximum computation done. However, in proof of stake there is no computation involved. It is just a cost less simulation. A byzantine slot leader may reuse its slot to extend multiple paths. Also as the slot election process is based on output of a random verifiable random function 4.3.1. It is possible that two or more nodes get output which is below the protocol threshold for a given slot. Thus all winning nodes will try to add a block to the slot. So forks are possible in the system. Ouroboros resolves this by considering forks into short forks and long forks by dividing the chain in regions in time domain in following way:

- **Short Fork**. When the fork is recent i.e when the chains diverge up to K blocks, follow the chain which is the longest.

- **Long Fork**. Follow the Plenitude rule i.e when the the chains diverges more than K blocks. Go back in time where the fork happened and isolate a short region after it within a certain time range. Count the number of blocks added to both chains and pick the one with more blocks. That is pick the chain which is more dense in the specific region where the fork started.

Double Spending Ouroboros Genesis effectively guards against double spending attacks, as all transactions are validated before those are added to the block. These validators i.e transaction endorsers get a portion of transaction fees proportional to number of transaction they verify. Transaction endorsers are linked with block, their validated transactions go into, similar to slot leader. **Transaction Finality** Transaction finality is very abrupt in Ouroboros Genesis. Once a transaction is added to the blockchain it will be finalized in approximately 5 minutes for a 0.10% adversarial stake. However, with current Bitcoin implementation for an adversary with hash power of 0.10% we require about 53 minutes (6 blocks) to be sure that transaction has been finalized and will stay in the chain [44].

4.3.3 Performance

Scalability Ouroboros Genesis have good network scalability similar to Nakamoto Consensus. It can work with any number of nodes added. However, currently its transactions

scalability is limited. It can perform approximately 100 - 150 transactions per second that is similar to PayPal and is much higher than Bitcoin with same level of security and network scalability. However, developers are working to enhance scalability using sharding and other mechanisms like side and off chains. However, those may impact the security of the system and reduce from threshold of 50% to 33%. Ouroboros Hydra [4, 29] will solve the transaction scalability issues of Ouroboros family. **Scalability Enhancements** Ouroboros upcoming version i.e Ouroboros Hydra will support Sharding, side-chains and off-chains for enhancing transaction throughput of the system [29].

4.3.4 Generic

Delegation of Mining At start of an epoch, participating nodes know whether those can mine blocks in some slots or not? If a node wins the lottery, it can go offline till the time it has to mine the blocks. Other than this it can also delegate the responsibility of mining to other nodes by assigning delegation rights to some other node in the network using the blockchain itself. Even a more simple approach to mine blocks on behalf of other nodes is to use proxy signature.[8] **Reward Mechanism — Incentive Structure** Reward mechanism of Ouroboros Genesis also differs from other proof of stake protocols. In other proof of stake protocols like peercoin [65] miners get rewarded by transaction fees for transaction those add to blocks. In Ouroboros Genesis besides slot leaders reward is also distributed amongst set of endorsing nodes, committee members those security party computation and nodes involved in sending messages to multi party computation protocol. Before adding a transaction sequence to the block, those have to be endorsed by the input endorsers. Endorsed sequence can then be added to any blocks for upcoming 2000 slots. The input endorsers are also linked to the slot in a similar fashion to slot leader. These input endorsers verify transactions to be added to the blockchain. Rewards are distributed at epoch level rather than block or slot level. This concept can be related to ByzCoin [67] where other than leader nodes who have recently produced blocks verify transactions. **Nash Equilibrium** Ouroboros Genesis is in Nash Equilibrium however bitcoin is not. **Smart Contracts** Ouroboros Genesis does not support smart contracts. However, it is on the time line for the Ouroboros family and will be supported by the newer versions probably the Ouroboros Hydra.

4.4 Comparison

The consensus in this class relies on some resource to win the lottery to generate the next block for the blockchain. The probability of winning is proportional to the resources invested in the network. Performance of the protocols depends on the computational puzzle involved. However, for protocols where no computationally hard puzzle is involved additional risks like nothing at stake are involved like Ouroboros family. Thus special mitigation measures have to be taken to cater for such attacks. Table 4.1 compares protocols from PoX Family. Bitcoin was the first protocol designed to operate in open settings where everyone can join or leave the network at will, and no pre-established identities are needed. It provided security guarantees, but in the trilemma of security, scalability and decentralization, it sacrificed transaction scalability. Changes have been suggested to improve upon transaction scalability. It included changes to current parameters like block size or changes in puzzle design. Different experiments on changing block size have shown that it can achieve maximum throughput when the block size is set to 4MB contrary to existing 1MB block size given constant block interval of 10 minutes [33]. Bitcoin-NG was designed to achieve same security and decentralization guarantees as Bitcoin and to enhance its throughput. Authors of Bitcoin-NG introduced the novel idea of key and micro blocks, but those failed to achieve any significant performance upgrade as they designed Bitcoin-NG to stay consistent with existing implementations of Bitcoin. Many ideas and protocols have been proposed to enhance the performance of Bitcoin after Bitcoin-NG. The majority agrees that virtual mining, like proof of stake, is the solution to get enhanced performance. However, proof of stake solutions had its drawbacks. The latest addition to proof of stake protocols is Ouroboros currently implemented in Cardano blockchain [26]. It has been designed from the start to provide security guarantees and decentralization same as Bitcoin yet achieve performance enhancements.

Parameters	Protocols		
	Nakamoto	Bitcoin-NG	Ouroboros
Adversay Tolerance	50% but practically it is 25% as with 25% or more adversary block finalization problems occur	50% for key blocks	50% stake
Consensus Finality	Eventual and Probabilistic	Eventual and Instant	Eventual and Probabilistic
Strong Consistency	No	No	No (Ouroboros Genesis provides consistency guarantees)
Forks Handling	Longest Chain	Longest Chain	Longest Chain
Block Production Time	approximately 10 minutes	~~100 seconds for key blocks;~~ variable for micro blocks	20 seconds
Puzzle Design	Computation Intensive	Computation Intensive and virtual	Virtual
Puzzle to Solve	SHA 256 hash with fixed number of zeros at start	For key blocks: SHA 256 hash with fixed number of zeros at start	Proof of stake
Design Goals	Security	Performance enhancement	Performance with Security guarantees

Operational Environment	Permission-less	Permission-less	Permission-less and Permissioned
Block Size	1MB,4MB	1MB	-
Transaction Propagation Model	Partially Synchronous	Partially Synchronous	Partially Synchronous
Throughput	3-7 tps	7 tps	100 -150 tps
Message Complexity	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Transaction Scalability	No	No	yes
Network Scalability	Yes	yes	yes
Smart Contract	Limited scripts	No	in Ouroboros Hydra
Availabiltiy of Code	Yes	No	yes
Realization	Bitcoin	Aeternity	Cardano

Table 4.1: A Comparison of PoX Protocols

Chapter 5

Consensus Based on Voting Scheme

Mostly for the vote based consensus algorithms, it is assumed that nodes in the network have either some type of pre-established identities associated with them or some public key infrastructure system is in place[55]. These protocols usually tolerate upto $\frac{1}{3}$ of faulty nodes in the network. There resilience against malicious nodes is increased with more nodes added to the network. However, at same time performance starts decreasing after a threshold due to increased number of message communication between the nodes [38] as a fully connected topology is need and all messages are multicasted to all the nodes.

5.1 Practical Byzantine Fault Tolerance

Practical Byzantine Fault Tolerance algorithm was introduced in 1999 and is the first protocol to optimally perform in byzantine environment. It provides a high performance byzantine state machine replications with low latency thus processing thousands of requests per second. Due to its high transaction scalability and fault tolerance in byzantine environment against $\frac{n-1}{3}$ nodes it has been implemented in several modern distributed systems including permissioned blockchains like Hyperledger Fabric and Corda. Practical Byzantine Fault Tolerance is also now used in permissionless environment in combination with other proof of X based consensus to enhance the transaction through put of those algorithms. However, where it scales to transaction throughput it doesn't scale to network due to heavy messaging between the network participants. Thus when nodes are increased beyond certain numbers, performance starts deteriorating due to bandwidth limitations[76].

5.1.1 Working

Concept Practical Byzantine Fault Tolerance benefits replication. If there is only a single server and it fails, users will not be able to access the resources. However, if the same data is replicated on different machines (servers) even if one is blinded out, other machines will render the requested information.

Algorithm implements a deterministic state machines replicated [95, 70]in a distributed

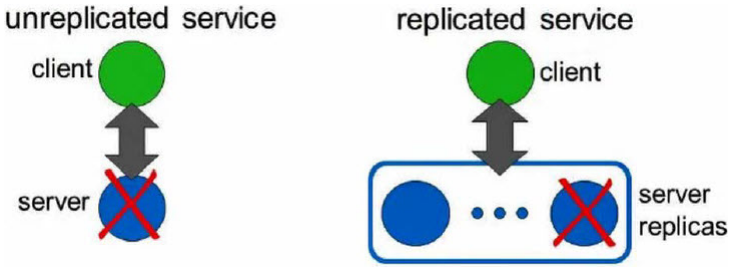


Figure 5.1: Practical Byzantine Fault Tolerance Benefits Replication

system across different nodes. All start at the same state, process same transactions in same order and correct nodes produce same results. Thus once a client node sends request to multiple trusted nodes in the network (eg servers) for certain information, if at least $f + 1$ nodes sends it back, client will accept the reply, f being the number of byzantine nodes. Algorithm works only in a partially synchronous environment with known bounds on message delays and delays between steps.

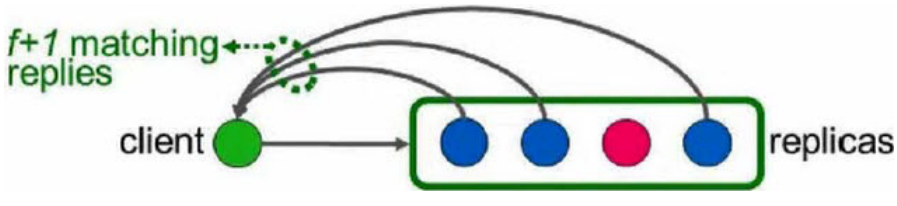


Figure 5.2: Practical Byzantine Fault Tolerance - Simplest Implementation

Working Nodes in the network implement the replicated deterministic service and are called replicas. These replicas are connected with each other over the network. Algorithm on these replicas (nodes), run in rounds called *View*, V. For each view one of the nodes implementing the replica acts as the view leader called *Leader or Primary* . View leader is selected deterministically. Other replicas in current view are called backups and are all deterministic as those implement a deterministic service. Views are sequentially numbered. Whenever, there is a view change (leader is changed) due to any reason, view sequence is incremented by 1.

For normal case operations, algorithm runs in 3 phases known as pre-prepare, prepare and commit phase. For normal case operations in pre-prepare phase, primary collects the request m from the client c and assigns a sequence number n to it. Sequence number increments sequentially and lasts over the lifetime of the algorithm. Primary signs the pre-prepare message and broadcasts it to all the replicas [28]. Backup replicas accept this pre-prepare

message only if it is in the current view v and its sequence number n is not repeating and is within a bound. In other words we can say that prepare phase ensures order within views. That is primary can not assign same sequence number n to two different requests in the same view. If this happens, backups can ask for a view change procedure. If some backup replica accepts the pre-prepare message, it will generate the prepare message using current view number v, message sequence number n, cryptographic digest $D(m)$of the message m and signs this prepare message. Each non byzantine backup replica who have accepted the pre-prepare message, will broadcast its prepare message to all the backups. All the backup replicas on receiving the prepare message will generate the commit message using current view v, message sequence number n, and cryptographic digest $D(m)$ for message m and will sign it. All non faulty replicas will then broadcast this commit message. Replicas will collect at least $2f + 1$ matching commits and will send reply to the client. Client will collect results from the replicas and returns results that appears in at least $f + 1$ replicas. Same has been summarized in Algoritm 4 and 5.3

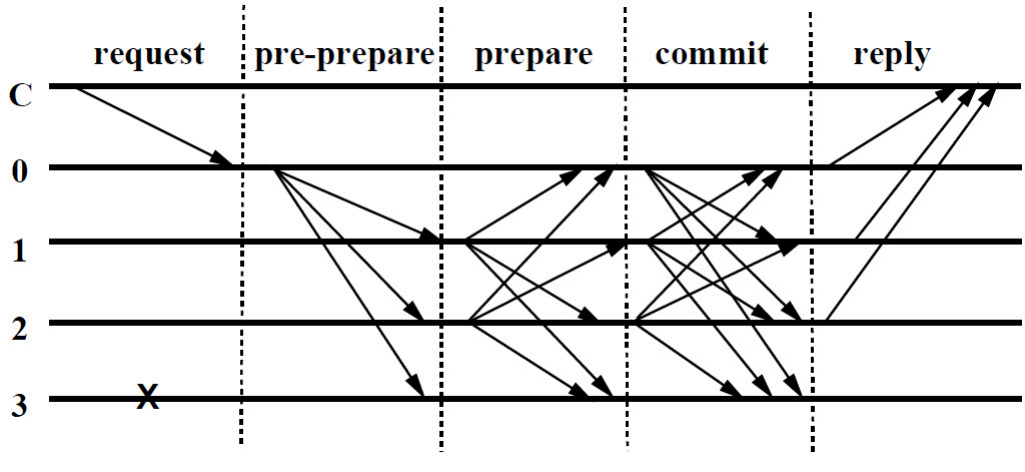


Figure 5.3: Practical Byzantine Fault Tolerance - Normal Case Operation

However, in case of time outs where a primary does not respond to backup replicas over a set interval or the backup replicas receive a message with sequence number n that is beyond the boundaries of current bound, a view change procedure is initiated. Next primary will be selected deterministically using formula $new\ primary\ =\ view\ number\ mod\ 3f + 1$. Again 3 phase operation we be run to finalize the commit for new view leader.

Algorithm 4: Practical Byzantine Fault Tolerance Algorithm - Normal Case Operation

```
1  Function Main():
2      NMp = Number of Message_prepare
3      f = byzantine nodes
       /* request stage                                   */
4      client sends a request to leader node
       /* phase 1:  pre-prepare stage                     */
5      leader relays the request to replicas
6      replicas update local state
       /* phase 2:  prepare phase                         */
7      relay Message_prepare to all replicas including leader
       /* phase 3:  commit phase                          */
8      if NMp ≥ 2f+1 then
9          execute client request
10         update local state
11     end
12     all replicas send reply to client
13 End Function
```

5.1.2 Security

Sybil Proof Practical Byzantine Fault Tolerance is not sybil proof. Nodes join and participate in the network based on their identities. Each identity has a voting power to accept or reject the transaction. Thus, if a node creates multiple identities, then that node can harm the system even with a single device. There has been few proposal and implementation where people have tried to avoid sybil attack using other mechanism like proof of work or proof of stake algorithms after a number of transaction. **Consensus Finality** Practical Byzantine Fault Tolerance offers instant finality. Once a consensus has reached on a transaction, it is final. There is no need to wait for some specific time or specific number of transaction to happen to get confirmation for earlier transactions. Thus once the nodes in the network have agreed on the message, it is secure and it will not be undone due to some other operations. **Forks** As there is absolute and instant finality of transactions there will never be forks. Nodes will agree to only one value and that will be treated as the true value, else nodes will abort the operation. Thus algorithm provides total ordering and liveness property for all transactions provided byzantine replicas are less than 33% to total [27]. **Established Identities** Algorithm requires that nodes establish identities before the protocol is run. These identities are used to sign the prepare and commit messages and in case of leader the pre-prepare mes-

sage as well. Identities can assigned using some public key infrastructure or fixed identities like Message authentication codes may be used. This nature of fixed identities, hindered its use in open public blockchains. But people have come up ideas where they assign those identities using proof of work or proof of stake mechanism and then use Practical Byzantine Fault Tolerance mechanism to reach consensus. [68] **Adversary Tolerance** Like most other byzantine fault tolerant systems, Practical Byzantine Fault Tolerance algorithm offers resilience against byzantine process provided $n > 3f + 1$ where n is the total nodes and f is the byzantine nodes, that is total replicas (nodes) in the network are more than 3 times the byzantine nodes.

5.1.3 Performance

Transaction Scalability Practical Byzantine Fault Tolerance provides best transaction scalability unless nodes in the network are kept below a threshold number and is only limited by the bandwidth of the primary (leader) node. On a pentium III with 8 bytes of argument size it can perform 15000+ operation per second with 100 replicas [28]. **Network Scalability and Message Complexity** In Practical Byzantine Fault Tolerance all replicas have to broadcast all messages. These broadcast messages create an overhead to network. If the nodes are increased beyond a threshold, transaction through put starts decreasing instead of increasing due to message communication involved. Thus it can be commented that this algorithm does not scale out with the network requirements. The message complexity of the algorithm is $O(n^2)$

5.1.4 Generic

Reward Distribution As in Practical Byzantine Fault Tolerance all the replicas participate in decision making process and add transaction by a final vote, thus all the reward related to the transaction fees is distributed amongst all the replicas. There is no reward variance for miners or replicas. **Resource Efficient** Unlike few PoX algorithms which require some type of resources to reach network consensus, Practical Byzantine Fault Tolerance achieves consensus without using any hard computation or resources. The algorithm in original puts minimal overhead on normal operation of the system it is deployed on.

5.2 BFT-SMART

BFT-SMaRt is a modular byzantine fault tolerant state machine replication / consensus library. It implements a deterministic state machine replication algorithm very similar to the Practical Byzantine Fault Tolerance 5.1. Library is open source and is written in Java language. It enhances throughput of byzantine fault tolerant consensus by multiple factor by benefiting parallelism of modern hardware, contrary to Practical Byzantine Fault Tolerance and UpRight [30], the two famous byzantine fault tolerant consensus algorithms. Also the BFT-SMaRt enables durable services by providing efficient and transparent support for replica set [71] reconfiguration. Nodes can be added and removed while those are actively participating in the network. This library can be also be configured to work in crash tolerant systems only rather byzantine environment where nodes may crash or behave maliciously. When consensus are sought in crash tolerant system, its performance is enhanced many folds..

5.2.1 Working

Like Practical Byzantine Fault Tolerance, BFT-SMaRt also runs in 3 phases namely *Propose*, *Write* and *Accept* phase. Protocol initiates with same initial state on all replicas and assumes that each node is connected to every other node on a trusted channel and network is partially synchronous. Consensus execution is triggered when a client multi casts a request to all the replicas. Each consensus instance is identified by a sequence number i and begins with leader (one of the replicas is selected as leader) proposing a number of request to be decided in that consensus round.

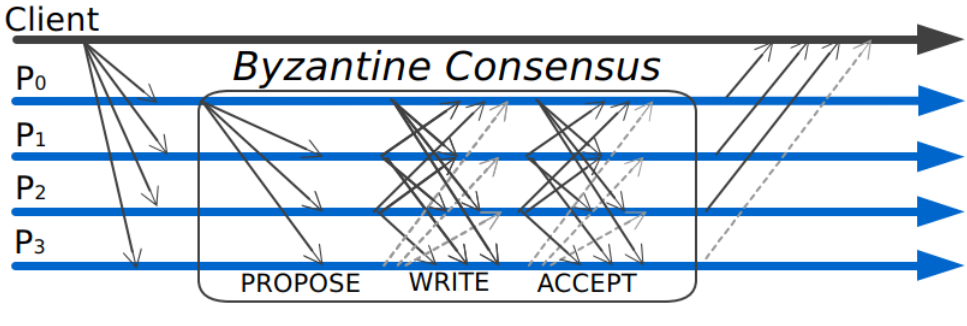


Figure 5.4: BFT SMART Normal Operation

Leader sends a *Propose* message to all the replicas containing aforementioned batch, multicasted by the client as shown in fig 5.4. All the nodes on receiving the *Propose* message,

validate the proposed the batch and verify if it has been sent by the leader for the current consensus. If these replicas successfully validate the proposed batch and verify the leader, they will register the proposed batch and multicast *Write* message to all the other replicas including leader the hash cryptographic hash of the proposed batch and not the actual messages itself. If any replica receives at least $\frac{n+f+1}{2}$ *Write* messages having the same hash, it will multicast an *Accept* message to all other replicas containing this hash. If any replica receives at least $\frac{n+f+1}{2}$ *Accept* messages having the same hash as of the proposed batch, it considers that as the decision for that consensus instance. Algorithm 5 depicts the same.

If during the propose phase, replicas are unable to verify the leader or if the proposed does not get validate then consensus do not occur and new leader is to be selected. New leader is elected in *synchronization phase* and all the replicas are forced to jump to the same consensus instance i.

5.2.2 Security

Sybil Proof This protocol is not sybil proof. Nodes join and participate in the network based on their identities. Each identity has equal voting power to accept or reject the transaction. Thus, if a node creates multiple identities, then that node can harm the system even with a single device. **Adversary Tolerance** BFT-SMaRt behaves different for byzantine faults and crash faults. It requires $n \geq 3f + 1$ replicas to tolerate upto f byzantine faults to ensure correctness. However, if system is configured for crash faults only then it requires $n \geq 2f + 1$ replicas to tolerate f crash faults. Moreover, as the system supports reconfiguration, it is possible on the fly to change the n and f through *leave* and *join* operations. Under all the configurations, system requires that reliable trusted point to point communication channel exists between all the nodes. System implements these channels using message authentication code over TCP/IP. **Liveness**BFT-SMaRt works in an eventual synchrony environment.Thus liveness property will always be achieved i.e committees will reach consensus in polynomial time. **Forks** Protocol achieves absolute and instant finality of transactions in less than a second, thus there will never be forks. Nodes will agree to only one value and that will be treated as the true value, else nodes will abort the operation. Thus algorithm provides total ordering and liveness property for all transactions provided byzantine replicas are less than $3f + 1$ to total. **Consensus Finality** BFT-SMaRt offers instant finality. Once

Algorithm 5: BFT SMART Normal Operation

Function `Main()`**:**
- NMw = Number of $Message_{write}$
- NMa = Number of $Message_{accept}$
- f = byzantine nodes
- n = total number of replicas

`/* request stage */`

- client multicasts a request to all replicas

`/* phase 1: propose stage */`

- leader assigns a consensus instance number i to each request
- leader relays requests $request$ to replicas and i
- replicas validate $request$ forwarded by leader
- **if** $request$ *is not validated* **then**
  - exit current consensus instance i
- **end**
- register the $request$
- update local state

`/* phase 2: Write phase */`

- multicast $Message_{write}|Hash(request)$ to all replicas including leader

`/* phase 3: Accept phase */`

- **if** $NMw \geq \frac{n+f+1}{2}$ **then**
  - multicast $Message_{accept}$
- **end**

`/* phase 4: commit phase */`

- **if** $NMa \geq \frac{n+f+1}{2}$ **then**
  - execute client request
  - update local state
- **end**

End Function

a consensus has reached on a transaction, it is final. **Established Identities** Nodes (replicas) have well established identities. Each replica have symmetric key for replica-replica communication and for replica-client communication. These symmetric keys for replica-replica communication are generated for each replica through Signed Diffie-Hellman using RSA keys and for replica-client communication using identities of the end points.

5.2.3 Performance

Communication Complexity Algorithm is designed in such a way that each node is to communicate with every other node present in the network. This requires substantial communication amongst the nodes. The communication complexity of the entire network becomes almost $O(n * n)$ or $O(n^2)$). **Transaction Scalability** Algorithm is designed to provide high transaction throughput in both byzantine and crash failure environments. It performs bet-

ter in crash failure environment than byzantine environment. The final throughput depends on multiple factors like number of cores of machine on which library is deployed, message length and number of byzantine nodes. Experimental results have shown that BFT-SMaRt can achieve throughput of around 80 thousand transactions per second in a LAN [97] and has a very low latency overhead for WAN [98]. **Network Scalability** BFT-SMaRt provides better network scalability than Practical Byzantine Fault Tolerance algorithm. For BFT-SMaRt network becomes saturated at about 1000 participants where as for Practical Byzantine Fault Tolerance it saturates at about 100 nodes [97]. This is due to communication overhead that increases with increasing number of participant in the network.

5.2.4 Generic

Contrary to all other byzantine fault tolerant consensus protocols based on replica, BFT-SMaRt offers dynamic system that can grow or shrink on the fly. Protocol enables addition or removal of the replicas during active operation. This can only be done by system administrator by running *View Manager* client.ss BFT-SMaRt is the most widely tested and the most advanced implementation amongst byzantine fault tolerant consensus [25]. It is implemented inside HyperLedger Fabric [55], Corda [25] and Symbiont.

5.3 Algorand

Algorand has been designed to overcome the shortcomings present in Bitcoin 4.1 and make operations energy efficient. It introduces a novel byzantine fault tolerant algorithm titles BA* to reach consensus and achieve network scalability. It couples verifiable random function and BA* for selection of committee members and leader for block creation. In the resultant system, transactions are confirmed almost instantly in less than a minute, blocks are produced with very low latency and possibility of forks is almost negligible. Like Bitcoin, Algorand has been designed to operate in open settings where nodes are free to join and leave at random, and it achieves true decentralization, as its design guards against pooling of nodes and delegation of tasks to other nodes.

5.3.1 Working

Concept Protocol runs in rounds. In each round it selects randomly a leader and few committee members. Leader adds transactions to the block and committee members verify the block. Committee members who validate the block, sign the block. If at least $\frac{2}{3}$ committee

members have signed the block, then it is propagated to the network. If due to some reason, committee does not reach consensus, it propagates an empty block to network and wait for next round to reach consensus. **Working** Protocol runs in rounds and in each round there are two distinctive steps. In first step a leader to propose new block for the current round and committee members to validate the block are selected. And in second round, protocol achieve consensus over the proposed block.

Random selection of leader and committee members is done through cryptographic sortition. Every network participant has a public and private key pair (pk_i, sk_i) associated with itself. When the round stars, every network participant tries to select itself as a leader or committee member based on proportion of its investment in the system performing *cryptographic sortition* by computing a hash and a proof π instituting a verifiable random function (VRF)

$$< hash, \pi > \leftarrow VRF_{sk}(seed||role)$$

Sortition consumes node's private key sk_i, publicly known random value *seed*, a *threshold* describing expected number of users for current role usually between 20 to 70 for leader selection and 2000 for committee size provided there are 80% honest nodes in network, *role* parameter distinguishing different roles that user can be selected for, stake of the node in the system*w*, and weight of all the stakes present in the system *W* using following algorithm.

$$< hash, \pi, j > \leftarrow Sortition(sk_i, seed, threshold, role, w, W)$$

Verifiable random function runs locally on every node. If a participant gets the hash below a certain threshold, it gets selected for *j* sub-roles. Other nodes can verify the selected node's claim using *VerifySort* routine.

$$< j > \leftarrow VerifySort(pk_i, hash, \pi, seed, threshold, role, w, W)$$

For every round *r* seed changes and is calculated using a verifiable random function. It is published at start of the round. It is derived from the seed of the last round *r-1* and is refreshed ever *R* rounds where R is decided by the system.

Once a node has selected itself as the leader for block generation, it packs the transaction in block. But it may be the case that some other node has also selected as leader as well.

Thus to minimize the unnecessary block transmission, Algorand discard the messages that do not have highest priority seen by the user so far by making leader node to wait for a time that is sum of time taken for last BA* consensus and time taken for gossiping the priority. Priority for a node can be calculated by hashing the hash output of verifiable random function concatenated with sub user index obtained in sortition. The message with highest priority gets propagated to the network.

After the block has been propagated in the network, protocol starts working on byzantine agreement using BA* protocol. BA* runs in two phases. In first phase, called the reduction phase the problem of agreeing on a block is reduced to one of the two option namely, final consensus or tentative consensus. In second phase, either proposed or empty block is appended to the chain.

In first phase, if at least $\frac{2}{3}$ of the committee members validate the proposed block, it final consensus is achieved and block is gossiped to the network for nodes to append the block to their copy of the chains. It is pertinent to mention here that Algorand ensures validity of the block and in BA* committee members only vote for hashes of block instead of its content. However, if due to any reason, less than $\frac{2}{3}$ validate the block, then a tentative consensus is achieved and block is marked for the next round. As in next round committee members get changed due to changing seed, new committee members will decide for the block. This process continues for a specified number of rounds (usually upto 8 rounds). After which if block is still not added to the chain, system will add an empty block to the chain. It has been depicted in Algorithm 6. However, there will be very rare such cases as protocol is designed to work effectively under strong synchronous assumptions and believes that after a period of weak synchrony, there will be a period of strong synchrony where nodes will reach consensus.

As seed changes for every round, different committee members are selected for every round.

5.3.2 Security

Sybil Proof Byzantine Agreement (BA*) instituted by the Algorand for the consensus require a leader for block proposal and few members for the committee to validate the block. Probability that a node will be selected as leader for block proposal or committee mem-

Algorithm 6: Algorand Algorithm

Input: A set of valid transactions and hash of previous block
Output: A valid block or an empty block

```
Join the Blockchain Network
Invest your assets in the system
Function Main():
    Blocklist = empty
    if Bk is not found then
        update seed
        update stake state
        Each node to elect itself as Committee Member or Leader
        if Elected as leader then
            fetch transactions
            construct block
            calculate message priority
            if message priority is highest then
                gossip block to neighbors
                add block to Blocklist
            else
                donot gossip block
            end
        end
        /* obtain consenesus for proposed block                    */
        run ReachConsensus(Blocklist)
    end
End Function
Function ReachConsensus(Blocklist):
    counter = 0
    while Blocklist is not empty or counter <9 do
        counter = counter + 1
        Block B = pop(Blocklist)
        ValicationScore = committee memebers to validate hash of B
        if ValicationScore ≥ 2/3 of total members then
            select longest chain
            add block to this chain
        else
            add B to Blocklist
        end
    end
    if counter ≥ 9 then
        remove items from Blocklist with index greater than 9
    end
End Function
```

bers is dependent on money it has in the system. Thus unlike other byzantine fault tolerant protocols, actual stakes in the system, rather than the pseudonyms matter here. **Consensus**

Finality Theoretically, Algorand achieves probabilistic finality. But time for the confirmation is less than a minute. **Forks** Even though Algorand offers probabilistic finality, yet in presence of strong synchrony and up to $\frac{1}{3}$ adversarial power, it guarantees a negligible probability for forks ($\approx \frac{1}{10^{18}}$) [56], thus allowing protocol to reach a definitive agreement on each block. In case of weak synchrony, short term forks are possible but Algorand uses BA* protocol to reach on agreement to resolve the forks. Thus, all the transactions are eventually finalized once system is in strong synchrony. The protocol priorities consistency over availability. If the nodes fail to reach consensus, it will add an empty block to chain and will try to reach consensus in the next round [43]. **Denial of Service Resistant** Algorand is Denial of Service resistant. Every node finds out locally in private whether it has been elected as member of the committee for the current round or not using cryptographic sortition? If it is the elected node, it will cast its vote on its turn. By the time adversary will deny this node, it would have posted its vote. Thus the algorithm is resistant against targeted Denial of Service attack. In case, adversary gets control of few nodes who have been elected as committee members, BA* handles it using temporary consensus. Algorand assumes that it will be very un-likely that majority in the network dishonest and rules out Denial of Service at entire network level [16]. **Double Spending** As the transaction are finalized in less than a minute and for every round committee members are changed who verify the blocks, double spending is not possible.

5.3.3 Performance

Network Scalability Protocol is highly scalable. New nodes can join the network without much impacting on the performance. Nodes are arranged randomly in graphs. Graphs are arrange in such a fashion that new nodes do not deteriorate performance. **Transaction Throughput** Algorand produces a very high throughput compared to Nakamoto consensus. It achieves 875 transactions per second [16]. **Latency for Block Propagation** It takes about 20 seconds for the block to propagate the network for one round. However, latency increases with block size. With a 10 MB block, latency for block propagation reaches to 50 seconds for one round. **Communication Complexity** Algorand uses gossip protocol for the communication amongst nodes. Each node is randomly connected to random peers making a random network graph. For each user, communication cost depends on the number of block

leaders and size of validating committee. As more users join the network, messages delivery takes long in the gossip network. In Algorand each node is part of at least one of the connected component in the graph. Message dissemination grows logarithmically with number of users in the connected component.

5.3.4 Generic

Incentive Mechanism Algorand does not define any mechanism for reward distribution [43].

5.4 Comparison

The consensus in this class relies on a voting mechanism to achieve consensus on true value. All the data is replicated to all the server or trustee nodes. Once the leader asks them to validate a transaction, these nodes then vote for the fact. If the value proposed (transaction) is consistent with the state held with the server and also satisfy protocol rules, nodes validate the transaction else it is rejected. Based on the number of votes received from the trustees or replication machines or servers, the transaction proposing node i.e. the leader decides whether to process the transaction or not. If processed, trustees or the servers update their state. Table 5.1 compares protocols from this Family.

Parameters	Protocols		
	PBFT	BFT-SMaRt	Algorand
Adversay Tolerance	33%	33%	honest nodes in consensus committee must be more than $\frac{2}{3}$
Consensus Finality	Absolute and Instant	Absolute and Instant	Probabilistic
Strong Consistency	yes	yes	yes

Forks Handling	Inhibits forks	Inhibits forks	Possible but mitigation for inhibition exists. Thus, practically forks may not exist.
Block Production Time	Instant	Instant	$<$ a minute
Block Size	-	-	512KB, 1MB, 2MB, 4MB,8MB, 10MB
Puzzle Design	Vote based	Vote based	Vote Based
Design Goals	Handle Byzantine Faults effectively	Handle Byzantine Faults effectively	Scalability
Operational Environment	Permissioned	Permissioned	Permission-less and Permissioned
Transaction Propagation Model	Asynchronous	Eventually Synchronous	Partially Synchronous
Throughput	15000 + tps	upto 10000 tps	875 tps with 10MB block size
Message Complexity	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$
Transaction Scalability	Yes	Yes	Yes
Network Scalability	No	No	Yes
Smart Contract	-	-	-
Availabiltiy of Code	Yes	Yes	Yes
Realization	Many (famous: Hyperledger/-sawtooth)	Few	Algorand

Table 5.1: A Comparison of Consensus Protocols Based on Vote Methodology

Both Practical Byzantine Fault Tolerance and BFT-SMaRt scale for transaction through

put upto a limit and then with addition of more nodes to network their performance starts degrading due to increasing message complexity. However, Algorand scales for both network and transactions. It relies on gossip protocol for inter node communication and it message communication only depends on immediately connected nodes.

Chapter 6

Hybrid Conensus Family

6.1 Elastico

Elastico has been designed for the permissionless blockchains with scalability in mind, and it scales linearly with the number of nodes in the network. Elastico benefits the idea that each transaction is verified by a small number of nodes i.e. a subset of total nodes, and there are many subsets of nodes verifying each transaction so that it remains secure. This concept allows processing many transactions in parallel. **Shards, Parallelization and Elastico** Sharding allows division of transactions into groups, thus enabling parallel processing. This concept is not new where identities are pre-established by some means like public key infrastructure or through the existing network infrastructure [31] and has been used for scaling SQL databases. However, for the identity-less system as in case of permissionless blockchains, this concept was first introduced by Loi et all in their paper "A Secure Sharding Protocol for the Open Blockchains" [76] and titled the suggested algorithm as "Elastico." **Elastico Working** Authors implemented the Elastico on top of Bitcoin. Elastico divides the nodes on the network into committees. Each committee then processes a disjoint subset of transactions called shards rather than the entire set of transactions. Internally, each committee uses classic byzantine fault-tolerant consensus protocol e.g PolyByz [77], Practical Byzantine Fault Tolerance (PBFT) 5.1 or SybilSensus [48]. However, these traditional Byzantine Fault Tolerance consensus requires pre-established identities. These identities are created at the start of each epoch by each processor using its IP address, a public key, and proof of work solution. These identities are then communicated to all the nodes comprising the constituent committee. Once all the nodes in the committee have reached consensus internally using Practical Byzantine Fault Tolerance protocol, each committee sends its values to a final committee called consensus committee. The final committee can use some existing byzantine fault-tolerant consensus protocol to reach consensus. The final value is then broadcast to all nodes on the network by this committee. The consensus committee also runs

a distributed commit and xor scheme for generation of an exponentially biased bounded set of random values to be used in the proof of work puzzle for the next epoch and is called *epochRandomness*. Algorithm 7 summarizes the general procedure for the protocol.

Algorithm 7: Elastico Algorithm

Input: A set of valid transactions
Output: A valid block

```
/* Join the Network                                                    */
Join the Blockchain Network as a Full Node (Mining Node)
Function Main():
    S = number of shards
    Committee Size = 2^s
    start epoch
    broadcast epochRandomness // randomness from last round for unique
       Proof of work solution
    run GenerateIdentities(epochRandomness)
    currentCommittee = create Committee
    while total committees ¡ s do
        while members in current committee < committee size do
            add members to current comittee
        end
        currentCommittee = new committee
    end
    form finalization committee with specific LSB in identity string
    reach consensus in shards using PBFT or any classical BFT mechanism
    reach final consensus using finalization committee
End Function
Function GenerateIdentities(epochRandomness):
    id=null
    while true do
        id=Hash(epochRandomness—IP_node|Pk_node|nonce) /* Pk - Public Key;
           */

        /* D defines a predefined difficulty                           */
        if id < 2^(id−D) then
            break
        end
        increment nonce
    end
    return id
End Function
```

6.1.1 Security

Elastico was designed with scaling out in mind, and it does that well. However, it sacrifices security to achieve decentralization and scaling out. It can guard against a maximum of 25%

faulty or dead nodes. It is pertinent to mention that it treats stopped or dead nodes as faulty nodes. **Sybil Proof**Elastico has been designed to work in open, permissionless blockchains where neither identities are pre-established, nor some type of public key infrastructure is in place [81]. Each node establishes its identity at the start of the epoch using its IP address, a public key and proof of work solution (Proof of Stake and Proof of Storage can also be used instead of proof of work) by searching a valid nonce to satisfy the following condition:

$$\mathcal{O} = \mathcal{H}(epochRandomness|IP|PublicKey|nonce) < 2^{\gamma - \mathcal{D}}$$

where $\mathcal{O}$ is a valid identity, $\mathcal{H}$ is a cryptographic pre-image resistant hash function, epochRandomness is a random value generated by final committee at the end of last epoch, and $\mathcal{D}$ determines work to be done by the processor to solve the proof of work puzzle and is predefined.

As the creation of identities involves computationally hard proof of work puzzle, identities can only be created proportional to the computational power. **Adversary Tolerance**Elastico can produce accurate results in the presence of $\frac{1}{4}$ byzantine or adversary computational power of the total computational power in the network. This must hold true for the committees as well. Inside a committee, there must be at least $\frac{2}{3}$ or 66% honest nodes of the total nodes in the committee i.e., each committee must have an honest majority with high probability. Also, Elastico treats failed nodes as Byzantine nodes. If the total failed or byzantine nodes in a committee exceed $\frac{1}{3}$, it will not guarantee correct results. **Liveness**Elastico works in synchronous or partially synchronous networks. Thus liveness property will always be achieved i.e. committees will reach consensus in polynomial time.

6.1.2 Performance

Scalability Elastico scales linearly to the computational capacity in the network because of parallelization of processing power and decoupling of the consensus process from the block data broadcasts, allowing an enhanced transaction rate. Contrary to tradition byzantine fault tolerant consensus it does not require a quadratic number of messages for committees to reach consensus rather uses a small number of broadcasts. This reduces message complexity to only $\mathcal{O}(nc^3)$ where n is total number of nodes and c is the number of committees. Experiments have shown that Elastico can scale up in magnitude by order of 4 when deployed

in scale to the current Bitcoin network. Following figure 6.1from Elastico white paper [76] depicts the relationship between network size, blocks produced and time consumed. **Band-**

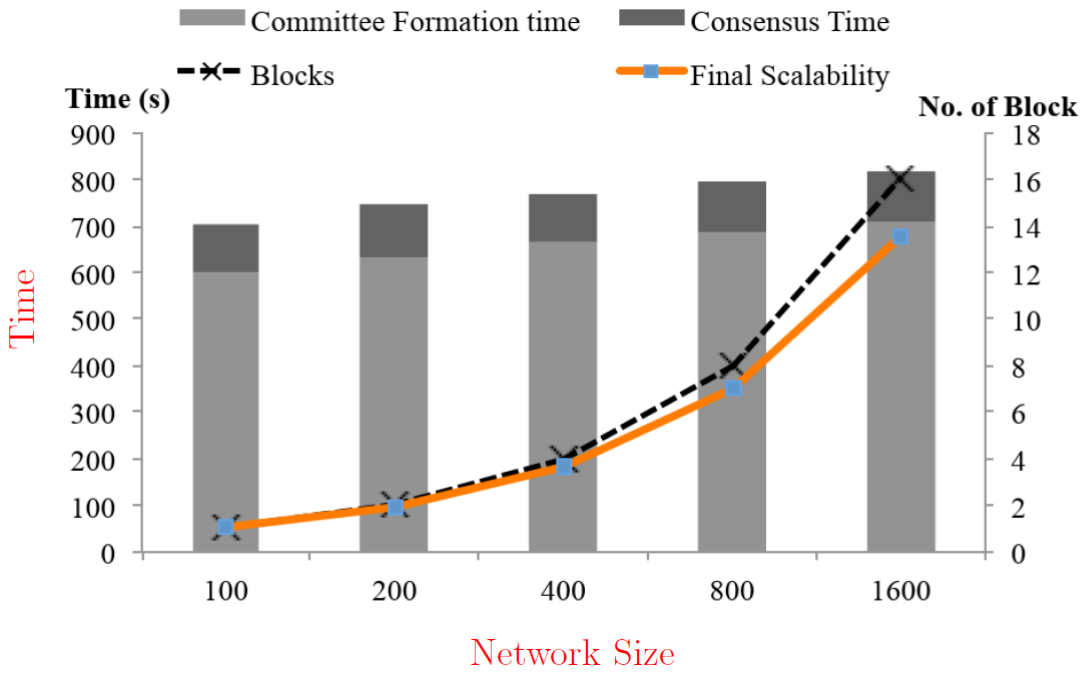


Figure 6.1: Elastico's Linear Scalability

width Requirement Contrary to some other consensus protocols like Bitcoin-NG where bandwidth per node is dependent on the production of key and micro blocks, the bandwidth of each node in the Elastico network is about 5 MB with little fluctuations. **Block Verification** A new block from one committee needs not to be verified by nodes of other committees. Instead of nodes only check if the particular block has been appended to the chain by Consensus Committee or not? Why so? Because the respective committee first verifies each block and then sent to the consensus committee. Block is only added to the chain once the consensus committee has verified it. This reduces local computation by the nodes, thus improving efficiency. Validators switch shards for each epoch. This requires that each validator stores complete transaction history i.e the global state of the chain. This at one hand improves security by providing a strong defense against adaptive adversaries but at the same time hinders performance [68]. The probability that the system will fail increases with an increasing number of shards in the network.

6.1.3 Generic

Elastico is fully compatible with existing data structure employed by the most famous blockchain, Bitcoin. However, one significant difference is Bitcoin is designed to work in an asynchronous environment where Elastico is designed to work in synchronous or partially synchronous environments. *Atomicity Across Shards Elastico fails to ensure atomicity of transaction across shards — this result in locking of funds forever in one shard if another shard rejects its transactions [68].

6.2 OmniLedger

OmniLedger relates closely to Elastico 6.1 and resolves many challenges left unsolved by the said protocol. It has completely revamped consensus for the permissionless blockchain environment and has been designed with following factors in mind:

Security
- Full decentralization
- Shard Robustness
- Secure Transactions

Performance
- Scale out
- Low Storage
- Low Latency

Contrary to existing blockchains where all the members of a consensus group or validators redundantly process and validate all the transactions, OmniLedger reduces this processing and validation load by splitting validators into shards.Each shard holds its own transaction ledger i.e shards have their own sub blockchain. Other than sub-chains for shards their is a global chain called identity chain that stores identities of all validators. Transactions across the shards are completely atomic and are handled by Atomix protocol introduced in OmniLedger consensus design [68]. OmniLedger builds on top of ByzCoin 6.3 , Hybrid Consensus, Ouroboros 4.3 and Algorand 5.3.

6.2.1 OmniLedger Working

Concept OmniLedger like Elastico 6.1 benefits sharding. Nodes in the network are grouped into different shards. Each shard validates and processes a unique and different sub set of

transactions. This enables parallelization thus allowing OmniLedger to scale out with improving the system throughout with a factor of number of shards. **Working** OmniLedger network consist of validating nodes termed validators. Each validator (i) is identified by a public key and has a public private key pair (pk_i, sk_i). The system runs for some fixed time unit called epoch. An epoch can be an hour or a day or any time, as determined by the system. To participate in the network for an epoch *e*, validator are required to establish an identity in epoch $e-1$. Identity can be established through a Sybil attack resistant mechanism like proof of work puzzle solution or proof of stake [74]. These identities are then published to a global identities chain. Identity chain is other than the transactions chain. Validators are then evenly assigned to shards at the start of epoch *e* using RandHound [100] protocol ensuring validators in a shard are a sample of the overall system. RandHound ensures distributed randomness, unbiasability and unpredictablitiy but requires a leader node to orchestrate protocol run. This leader can be some fixed third party trusted node or a node elected through a deterministic way. However, OmniLedger makes use of verifiable random function (VRF) based approach for leader election [80]. A verifiable random function (VRF) comsumes x as input and produces a random hash of a fixed length ($l-bits$) and a verifiable proof π based on private key sk_i. Each validator (*i*)computes a ticket ($t_{i,e,v}$)at start of epoch *e* using concatenation of configuration file ($config_e$) containing all valid registered validators on identity blockchain for the current epoch *e*, and view counter (*v*) passed to verifiable random function (VRF) i.e $t_{i,e,v} = VRF_{sk_i}(\text{"}leader\text{"}||config_e||v)$. Tickets thus computed are gossiped amongst other nodes in the network. After a known maximum time Δ in minutes, validators lock the valid ticket with lowest value appeared thus far. Node who owns this ticket is accepted as the RandHound leader for the current epoch e. If for any reason, within a time Δ elected leader fails to start the RandHound protocol, then current run is marked as failed by the validators. Next, view is incremented by 1 and entire lottery process for the leader election is repeated. Now, the RandHound protocol is run with elected leader. After successful completion of the protocol run, leader broadcasts random value for the current epoch rnd_e and correctness of proof. Now all the properly registered validators who are part of the network for current epoch fist verify rnd_e and then use this random value to calculate permutation π_e of each validator. Once permutation calculation is done, it is used to assign validators to shards by dividing validators in m equal sized buckets [68]. It is pertinent to

mention that for each epoch OmniLedger only swaps a portion of validators in shard. This makes it possible for the system to remain functional during epoch transitions as well. To maintain the current system state, it uses unspent transaction output (UTXO) model [88]. UTXO model allows full nodes to drive the current state of the distributed ledger from the blockchain itself. At time of boot strapping, a full node crawls the entire blockchain and builds database of all valid UTXOs. For cross shard transactions, OmniLedger uses Atomix along with UTXOs model. Inside a shard nodes reach consensus using ByzCoinX protocol. It ia a modified version of Practical Byzantine Fault Tolerance protocol with modified messaging propagation mechanism. The adopted messaging mechanism resembles closely to two-level tree. As we know that before start of the epoch a randomness rnd_e was generated by the leader for validators to assign shards. Same randomness is used by epoch leader to assign nodes to groups within a shard. The leader also randomly selects a group leader from the group at start of the ByzCoinX protocol roundtrip. It is the responsibility of the selected group leader to manage communication between the protocol leader and the group members. In case the selected group leader fails to respond for a predefined time, the protocol leader selects another group leader randomly.

6.2.2 Security

Sybil Proof OmniLedger is designed for open permissionless blockchains in absence of pre-established identities and public key infrastructure. Nodes on joining the network establish their identities using any Sybil attack resistant mechanism like proof of work or proof of stake. These are then published to a global identity chain. **Adversary Tolerance** OmniLedger can produce true results i.e it guarantees liveness and safety in presence of $3f + 2$ nodes where f are the byzantine nodes [75]. If the worst cases happens i.e the network contains $\frac{1}{3}$ malicious nodes uniformly distributed across the entire network, then all remaining honest nodes have to respond for correct calculation of the result, else the protocol will fail. It ensures an absolute and instant consensus finality. Once a block gets added to the chain it can not be pruned. Therefore, all the work that is put for the validation of blocks is never lost which is contrary to other famous protocols like Bitcoin where due to forks sometimes entire blockchain branches are pruned. **Double Spending** There is a less than 10% probability that system will become vulnerable to double spending attacks.

6.2.3 Performance

Scalability OmniLedger scales out linearly with number of nodes in the network. Its throughput varies with number of nodes in the network, division of these nodes into number of shards, number of nodes in each shard and adversarial power. Fig 6.2 depicts the same for 1800 nodes in presence of varying number of shard sizes and adversarial powers.

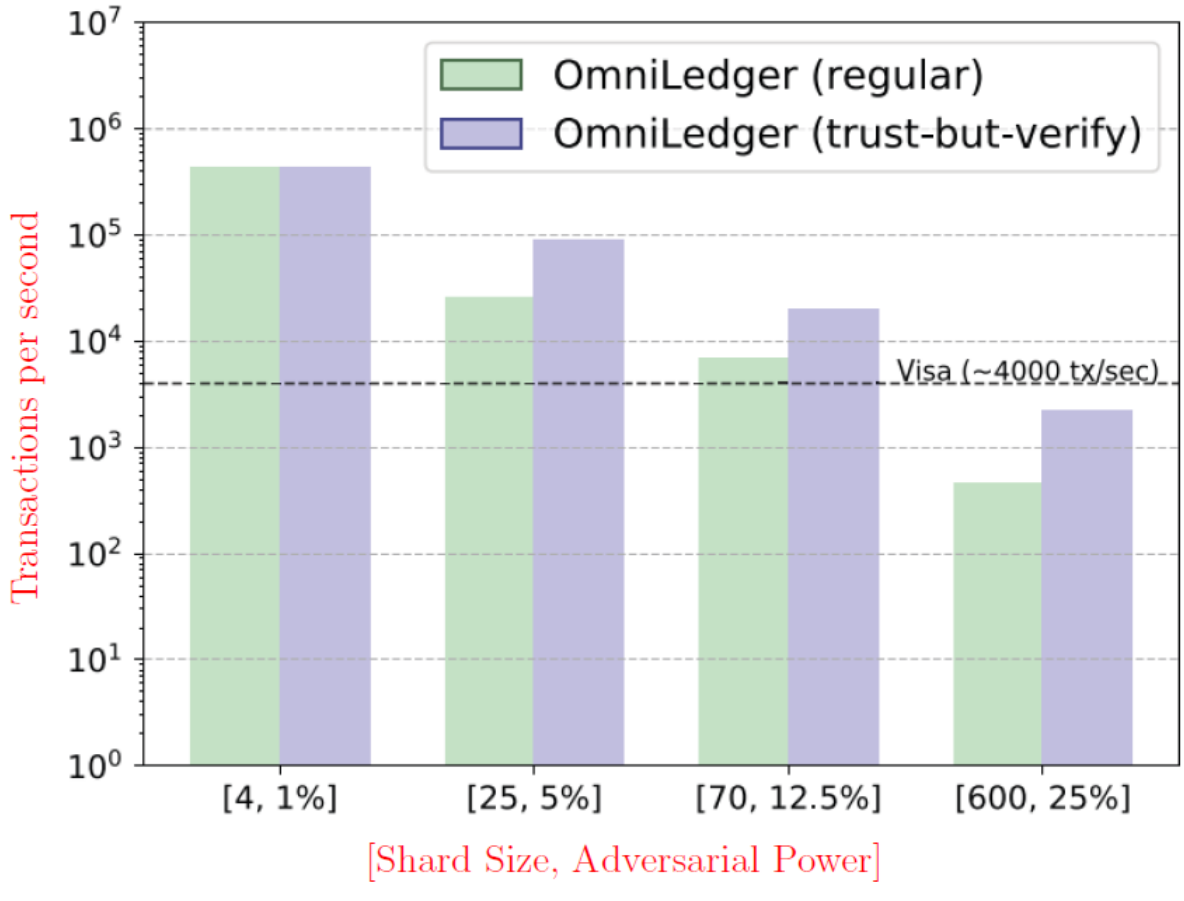


Figure 6.2: OmniLedger Throughput (Copied from [68])

It achieves throughput of 4000 transaction per second i.e average throughput for visa in presence of 25% adversary with only 4200 nodes divided in 7 shards. This number is much lesser than number of full nodes on Bitcoin network. Currently Bitcoin network have 9477 active full nodes [18]. **Latency** Latency in OmniLedger is effected by shard size. It is not much effected by the number of shards. For 7 shards with 70 nodes in each, latency is approximately 4 seconds. However,once shard size is increased, also increases latency. Latency increases rapidly when transactions are cross shards. It becomes worse when multiple shards are required to validate a transaction and output shard has to run a consensus. However, usually it is not required. **Communication Complexity** OmniLedger modifies the

Practical Byzantine Fault Tolerance algorithm to achieve a message complexity of $O(log\ n)$. Originally, Practical Byzantine Fault Tolerance has a message complexity of $O(n^2)$ .This reduction in message complexity is achieved through a tree oriented collective signing where epoch leader and group leaders from shards selected for current epoch are responsible for the communication of messages 6.2.1.

6.2.4 Generic

Smart Contract Support OmniLedger does not support smart contracts. It can be coupled with other existing frameworks to add the smart contract support.

6.3 ByzCoin

ByzCoin has been designed for open, permissionless blockchains with security and performance in mind. It introduces the concept of Practical Byzantine Fault Tolerance inside proof of work, hence a new consensus mechanism which is a combination of Practical Byzantine Fault Tolerance and proof of work. By doing so, it establishes a consensus mechanism that is byzantine fault-tolerant and provides strong consistency guarantees [67]. It scales out Practical Byzantine Fault Tolerance in terms of transaction throughput leveraging collective signing using Schnorr multi signatures or Boneh Lynn Shacham signature scheme and in terms of network size using proof of work, that is it adapts to open permissionless environment using proof of work puzzle solution.

6.3.1 Working

Concept ByzCoin establishes a scalable Practical Byzantine Fault Tolerance consensus leveraging collective signing scheme by arranging validators in a tree structure. The current set of validators is determined using proof of work within a rolling window. The latest winner of proof of work in that window acts as the leader of the validators and hence is placed at the root of the tree. Validators get a share proportional to their mining power.
Working ByzCoin utilizes byzantine fault-tolerant scheme Practical Byzantine Fault Tolerance for consensus. However, Practical Byzantine Fault Tolerance has been designed to operate in a federated environment where identities are already well established and a designated leader. A leader acts as moderator for the other nodes. It proposes blocks to other nodes called trustees in the network using all-to-all communication. All nodes use a message authentication code to authenticate each other on the network. The protocol runs in

three phases named pre-prepare, prepare and commit phase.

To make it work it the open, permissionless environment, ByzCoin utilizes Nakamoto proof of work. Proof of work allows electing trustees and leader to run Practical Byzantine Fault Tolerance protocol in a Sybil proof way. Also, it shifts from symmetric message authentication code to public key infrastructure where nodes have their public and private keys to authenticate each other derived using Elliptic Curve Ed25519 signatures. To limit the number of trustees, the concept of a sliding window of fixed time duration has been introduced in the ByzCoin protocol. The time duration for this sliding window can be fixed to a day, a week, or even a month based on security and performance requirements. When a miner mines a block using a computation-intensive proof of work hard puzzle [82], it gets added to the rolling window. Once a new miner after winning the puzzle gets added to the window, the member that have found the block earliest in current window gets deleted from the window. These members in the current window act as trustees. Thus the membership of the trustees is done through proof of work. Therefore in ByzCoin, proof of work is also termed as *proof of membership*.

ByzCoin protocol like Bitcoin-NG 4.2 decouples leader election process from transaction verification to enhance the transactions throughput. Thus, it introduces the concept of key and micro blocks. Key block contains a proof of membership and determines membership for the rolling window as discusses in the last paragraph. Latest validator node which has obtained the membership of the trustees can add multiple micro blocks to the blockchain until a new node solves the proof of work puzzle. This validator node that has been added to the window the last, acts as leader for other nodes in the window and is responsible to get all the transactions validated before adding those to the micro blocks. Protocol employees collective signing principle for the purpose[101]. Hence, the leader node collects transactions and distribute it to other trustees. Trustees can be arranged in a tree pattern or any other fashion. Those trustee nodes validate the transactions and send their response to the leader. The leader then computes a challenge based on trustees response and sends back to them for verification. If verified successfully, the new micro block is added to the blockchain by the leader. Share of mining new blocks is distributed among the current set of trustees proportional to their hashing power i.e the number of key blocks they have mined in the current rolling window. As seen in fig 6.3 (copied from [67]) yellow block has been recently mined,

so is acting as the leader for the current session. In current window, yellow has mined two key blocks, pink has mined two key blocks and blue has mined three key blocks of the total seven mined blocks, thus yellow and pink will get $\frac{2}{7}$ share each, and blue will get $\frac{3}{7}$ share. An abstract version of the ByzCoin protocol is presented in Algorithm 8.

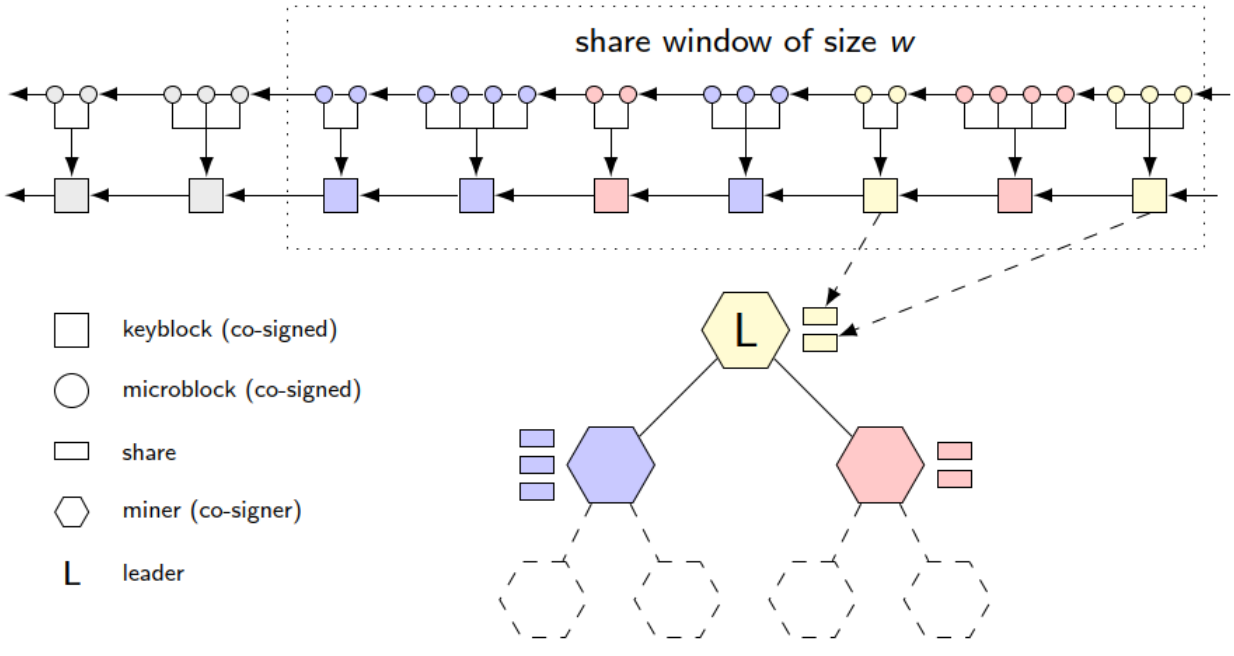


Figure 6.3: ByzCoin Design

6.3.2 Security

Sybil Proof ByzCoin is designed for open, permissionless blockchains in the absence of pre-established identities and public key infrastructure. Nodes on joining the network establish their identities using Nakamoto proof of work Sybil attack resistant mechanism. Thus total hashing power rather than the number of identities matter in the network. **Adversary Tolerance** ByzCoin can guarantee safety and liveness i.e it can produce correct blocks in the presence of $3f + 2$ nodes where f are the failed or byzantine nodes [67]. If the worst cases happen i.e., the trustees in sliding window constitute $\frac{1}{3}$ byzantine validating nodes, then all remaining honest nodes have to respond for correct calculation of the result, else the protocol will fail. Also, at least $2f + 1$ honest validating nodes must verify the micro block and an honest leader accepts it and adds it the blockchain. If the leader becomes byzantine, it will compromise the liveness property of the consensus. **Forks and Selfish Mining** Two protocols are running here. For micro blocks, there will be no forks, as Practical Byzantine Fault Tolerance protocol provides strong consistency [27] thus inhibits forking. However, key

Algorithm 8: ByzCoin Algorithm

Input: A set of valid transactions
Output: A valid block

```
/* Join the Network                                                    */
Join the Blockchain Network as a Full Node (Mining Node)
Function Main():
    s = fixed window size
    create Queue trustees of size s
    P = Hash of the last block
    Block_k = Key Block containing P, current GMT,coin base reward, target value
     and nonce
    Block_m = Micro Block containing ledger entries
    H = Hash function
    D = Difficulty level
    retrieve P
    while true do
        Construct Block_k
        /* will be true once key block has been found                  */
        if H(P,BK)<D then
            gossip Block_k to peers
            Dequeue(trustees)
            // remove node that found the block earliest.
            Enqueue(trustees,node) // node which has found the block
        end
        update nonce
        run PBFTModified()
    end
End Function
Function PBFTModified():
    leader = top(trustees) // node added to queue most recently
    leader packs transactions to a microblock Block_m sends Block_m to trustees
        // members of queue less top node
    trustees validate Block_m
    trustees send response Block_response to leader
    leader computes challenge Block_challenge for Block_response
    leader sends Block_challenge to trustees
    trustees computeBlock_ChRe against Block_challenge
    trustees send Block_ChRe to leader
    if Block_ChRe is verified then
        add Block_m to chain
    end
End Function
```

block generation may lead to forks or selfish mining like in Bitcoin [92]. However, protocol mitigates it by increasing entropy of the output deterministic prioritization function. This idea is to arrange the hashes of all the conflicting blocks in sorted order from low to high in

an array. Then compute the hash for this array. Now compute the modular using following formula. The remainder (final bits of the hash) will the block index for the winning block.

$$winning\_block\_index = arrary\_hash \; mod \; (arrary\_size - 1)$$

This resolves the issue of forks without any voting. **Attacks Envisaged** As the protocol is based on Nakamoto proof of work and Practical Byzantine Fault Tolerance protocol, it inherits strengths and weaknesses from both. The system is vulnerable to temporary Denial of service or slowdown attack owing to properties of the underlying byzantine fault-tolerant protocol. If the elected leader is malicious, it can hinder the consensus process by excluding minorities to take part in consensus. However, the protocol is less vulnerable to Eclipse attack as compared to Bitcoin as it is difficult for the malicious elected leader to convince the validators into accepting alternate transaction history that is controlled by the malicious attacker node. Practically speaking, an adversary needs to control at least $\frac{2}{3}$ of the validating nodes i.e. the voting power to launch the eclipse attack [78].

6.3.3 Performance

It improves performance in terms of both transaction scalability using Practical Byzantine Fault Tolerance protocol and network scalability using Nakamoto proof of work protocol. **Latency** The latency of the protocol almost remains fixed as a number of nodes participating in consensus does not vary much. Numbers will only vary when one node mines key blocks successively. However, the choice of the collective signature scheme used i.e. tree-based scheme or flat scheme and block size will impact the latency in the consensus. For a constant 144 validators, time to reach consensus for a block of 1 MB is about 10 seconds and for a 32 MB block, it is about 40 seconds for tree-based collective signing scheme. However, same rises to about 20 seconds and 110 seconds respectively for 1 MB and 32 MB block using flat-based collective signing scheme. **Communication Complexity** ByzCoin modifies the Practical Byzantine Fault Tolerance algorithm to reduce per round message complexity to $O(log\ n)$. Originally, Practical Byzantine Fault Tolerance has a message complexity of $O(n^2)$. This reduction in message complexity is achieved, employing scalable collective signing protocol. The collective signing also enables direct verification, thus enabling the system to reduce the signature verification complexity to a constant complexity of $O(1)$

from $O(n)$. Also reduces the typical case communication latency to $O(n)$ by using digital signatures instead of message authentication code. **Throughput** Authors have claimed that ByzCoin can perform 1000 transactions per second with 1 MB block (blocks size currently used by Bitcoin) when transactions are validated in a tree pattern. This is much higher than the throughput of Paypal witch is currently 193 transactions per second [53]. In case of overload, the system is designed to adapt by changing the micro block size to 2 MB. Also, if the tree pattern fails, the system adapts by following a flat pattern with 1 MB block size. With flat pattern system can process more than 70 transactions per second witch is almost 10 times higher than the current Nakamoto consensus.

6.3.4 Generic

Open Source and Code Availability This is an open-source project and code is available at GitHub at URL [34]. Code is written in Go Language.

6.4 Comparison

The consensus in this class relies on a Sybil proof mechanism for forming trustees group and a classical consensus mechanism like Practical Byzantine Fault Tolerance for achieving the consensus. Their performance mostly relies on the selection of protocols for both phases and arrangement of nodes in classical consensus for reaching a final consensus. Table 6.1 compares protocols from Hybrid Family.

Parameters	Protocols		
	Elastico	Omni Ledger	ByzCoin
Adversay Tolerance	25% faulty processes, can reach upto 33% if committees are *fully random*	upto $\frac{1}{4}$ of the computational capacity. Depends upon number of shards and members in each shard	25% for key blocks
Consensus Finality	Absolute and Instant	Absolute and Instant	Absolute and Instant

Strong Consistency	yes	yes	yes
Forks Handling	-	Inhibits forks	~~Inhibits forks~~
Block Production Time	~~10 minutes~~s	$<$ 1 minute	10 seconds for 1MB block and 90 seconds for 32MB block
Puzzle Design	Hybrid	Hybrid	Hybrid
Puzzle to Solve	PoW for committee formation And PBFT for consensus	Sybil proof for ~~committee formation~~ and classical for consenus	~~Sybil proof for committee formation and classical for consenus~~
Design Goals	Scalability	Scalability and Security	Scalability
Operational Environment	Permission-less	Permission-less and Permissioned	Permission-less and Permissioned
Block Size	1 MB	500KB, 1MB, 16MB	1MB upto 32 MB
Transaction Propagation Model	Partially Synchronous	Synchronous, internally to a committee	Partially Synchronous
Throughput	600 tps (depends on committee size and nodes / commitee)	upto 10000 tps	~~700~~ tps
Message Complexity	$\mathcal{O}(nc^3)$	$\mathcal{O}(n)$	~~$\mathcal{O}(1)$~~
Transaction Scalability	yes	yes	yes
Network Scalability	No	yes	yes

Smart Contract	-	No, can be modified to add support	-
Availabiltiy of Code	-	yes	yes
Realization	None	None	-

Table 6.1: A Comparison of Hybrid Family Protocols

These protocols have been designed to scale with network and adopt goods of both modern permission-less consensus like Nakamoto's Proof of work and classical consensus algorithms like Practical Byzantine Fault Tolerance. However, security and performance of such protocols mainly depend on how randomly those assign nodes to shards for Elastico and OmniLedger or a majority of honest nodes in consensus windows for ByzCoin. OmniLedger builds on top of Elastico and ByzCoin and tries to accommodate goods of both. Elastico's performance grows linearly with the number of nodes and shards [76] , but after a limit, it starts degrading due to heavy message load. Contrary to it, OmniLedger scales with network and number of transactions. Elastico's failure on an increasing number of nodes is inherited from Practical Byzantine Fault Tolerance. OmniLedger kills this issue by adapting to the concept of collective signing scheme adopted from ByzCoin. OmniLedger also improves upon the methodology of assigning nodes to shards. Elastico assigns nodes to shards based on a pattern of least significant bits in identity string, whereas OmniLedger makes use RandHound algorithm for assigning nodes to shards.

Chapter 7

Blockchain Consensus Protocol for IoT

7.1 Why Blockchain for IoT?

IoT devices have a widespread proliferation in our existing environment. Those find more and more autonomous deployments with every passing day. From small autonomous networks, we have grown to smart homes and smart cities and journey has just begun. However, to meet the true potential of autonomous behaviour, these devices (generally termed as nodes) need to interact and communicate with each other for exchange of data. For visualization of a digital world that truly represents the real world, we need a totally connected network of huge number of devices. And these devices have a peculiar characteristic, they generate huge amount of data. But, if few nodes become faulty or even worse if an adversary controls those. In such scenarios, different nodes on the network will propose different data to network. This breaks the trust on data proposed. Furthermore, data can be tampered or altered en route while it is travelling from one point to another. This worsens the trust on received data. Thus we need some mechanism, where the information received can be verified for its integrity.

Blockchain provides the most simplest and promising method for achieving this trust, in information received, in a distributed fashion. It provides transparency and allows all the nodes on the network to validate the data, that it has not been modified since its definition through cryptographic constructs. And it also helps devices to decide on a single true value in presence of malicious and adversary nodes using its consensus algorithms. But, blockchain has its own requirements for any system to adopt to it. Nodes need to have the following functionality:storage, routing, consensus and wallet services. Not all nodes have all the functionalities. Based on the role held by a node in a network, it can be categorized as Core Node, Consensus Node (Mining or Block generation node), Full Node, or Light Node. Table 7.1 summarizes different type of nodes and their functionalities [90].

Consensus capability is the most important for information integrity assurance. Consen-

Node Type	Storage	Routing	Block Generation	Wallet
Core Node	✓	✓	✓	✓
Consensus Node	✓	✓	✓	✗
Full Node	✓	✓	✗	✗
Light Node	✗	✓	✗	✓

Table 7.1: Node Types based on their Capability

sus algorithms enables untrusted nodes in the network to validate all the transactions and to conclude on a single universal truth in absence of central authorities. However, mostly the consensus mechanism those provide high security guarantees usually have a very high computational requirement besides other drawbacks like low transaction throughput and high latency. Contrary to requirement of such protocols, IoT devices are computationally very light and have very limited communication capabilities. Usually those are low powered embedded computational platforms with bare minimum capability to perform tasks. Besides this, those a limited storage, RAM and low powered communication mechanism like IEEE 802.15.4 [2]. Due to all these reasons, it is vital that only light weight consensus protocols be used with IoT or protocols be modified for use in IoT. During the current study, protocols were specifically studied for their suitability in blockchain with IoT devices as network participating nodes. For a blockchain to be deployed in IoT environment, it is desirable that it has features like decentralization, it should not be computation intensive as well as network intensive and it ought to have high network scalability. However, most light weight consensus rely on heavy communication among participants to maintain data integrity. This heavy communication some times become bottle neck in interference limited communication channels. Thus, mostly we have to struck a trade off between security and throughput. Other desirable features for such blockchains are high throughput and low latency. Table 7.2 summarizes protocols those can be used with IoT in any of the modes.

7.2 Consensus Algorithms for Blockchain for IoT

As table 7.2 hints upon, blockchain consensus stemming from proof of work or related hard problems does not suit well for IoT devices. Also, this gives the idea that blockchains for permission-less environments are not suited for IoT integration, because for nodes to participate in consensus process, those have to provide storage facility itself as indicated in table 7.1. Mostly, the IoT networks generate huge amount of data. It will not be possible for IoT

Consensus Protocol	Suitability	Remarks
Nakamoto Consensus	✗	Participate in Bitcoin network as a *Lite Node*
Bitcoin-NG	✗	Participate in Bitcoin network as a *Lite Node*
Ouroboros	✓	With external data storage service; *Hyperledger IROHA* employes PoS
PBFT	✓	Mostly existing blockchain platforms for IoT use PBFT protocol.
BFT-SMaRt	✓	-
Algorand BA*	✓	-
Elastico	✗	-
OmniLedger	✓	Provides very low storage overhead
ByzCoin	✗	Phase 1 protocol stems from Nakamoto consensus

Table 7.2: Suitability of various consensus protocols for IoT

devices taking part in consensus to keep the entire chain with this amount of data. Another issue of permission-less environment is that nodes require to sign and encrypt transactions using public and private keys. That makes it computation hungry compared to counter parts in permissioned environments where Message Authentication Code can do the task. In permissioned settings, a single entity or a consortium controls the blockchain rules and nodes have identities attached to them. This allows to use vote based high performance consensus protocols in permissioned environments. Hyperledger Family, Corda [22] and Tendermint [69] all rely on vote based algorithms to reach consensus. Those mechanisms have their own drawbacks associate with them as discussed in chapter 5 like high message complexity. But nodes can be arranged in different patterns to minimize bandwidth usage.

The three most suitable consensus algorithms for IoT devices are Practical Byzantine Fault Tolerance protocol 5.1 , Algorand 5.3 and OmniLedger 6.2. Pros and Cons of each are given below:-

7.2.1 Practical Byzantine Fault Tolerance

In Practical Byzantine Fault Tolerance all nodes have to participate in consensus process. It can tolerate upto $\frac{1}{4}$ byzantine nodes. That means if there is one malicious node, then at least 4 honest nodes are required to reach consensus on a truly proposed value else the system will fail to achieve termination property of consensus. It is not suited for permission-less blockchains but suits well for private and consortium blockchains. It has been adopted by

many permissioned chains like Hyperledger. It is characterised by a high throughput, low latency and low computational overhead. Its major pros and cons are given below:

Pros

- It is not computation intensive and network intensive (unless number of devices below few thousand).
- It has a very high transactions throughput.
- Network latency is low.
- Can tolerate upto 33% byzantine nodes.
- ~~Does guard against Sybil attack~~.

Cons

- It does not scale well beyond few thousand nodes after that its performance starts degrading.
- It can not operate in fully decentralized environments where identities are not established.
- Message storage over head is high.

7.2.2 OmniLedger

OmniLedger has been designed to operate in permission-less settings. It can tolerate upto $\frac{1}{4}$ adversarial nodes. It divides nodes on the network into shards using verifiable random functions. Within each shard it uses modified version ByzCoin Practical Byzantine Fault Tolerance. It has very low storage overhead as nodes are not required to store the entire chain [68]. Though using OmniLedger we can implement permission less blockchain for IoT but is poses a large communication overhead, which makes it an in efficient choice [115]. However, this is much efficient as compared to other permission less blockchain protocols.

Pros

- It can operate in both permissioned and permission less settings.

- It has a very network scalability.
- Guards against Sybil attack.
- Can tolerate upto 25% byzantine validating nodes.
- Transactions throughput is very high. Depends on number of nodes in each shard.
- Low storage overhead as validating nodes are not required to store entire transaction history.

Cons

- Computation overhead is much high as compared to Practical Byzantine Fault Tolerance protocol.
- It has a high message latency.

7.2.3 Algorand

Algorand can tolerate upto $\frac{1}{3}$ of byzantine computational power. It combines virtual mining technique proof of stake and its own vote based byzantine fault tolerant agreement mechanism BA* to achieve a high throughput system that is suited both for permissioned and permission-less blockchains. It provides ASIC resisability and inhibits centralization that is a desired property for good consensus. It also improves upon security, as nodes do not know in advance, which committee they will become part of [47].

Pros

- Well suited to operate in permissioned and permission-less settings.
- Guards against Sybil attack.
- It is highly decentralized.
- It also provides ASIC resisability.
- High network scalability.
- High transactions throughput.

Cons

- Throughput is limited as compared to Practical Byzantine Fault Tolerance but is very high compared to Bitcoin and Elastico.
- It has a high message latency, though less than OmniLedger.
- Nothing at stake attack is possible in Algorand.
- Usually there is no concept of money in IoT nodes. However, Algorand select nodes for consensus committee based on stakes node hold in the system.

7.2.4 Discussion

IoT devices have storage, computational and energy constraints. As discussed above it will not be possible for those devices to solve computationally hard puzzle. It can be concluded from 7.2 that if IoT devices have to be deployed in permissioned environment, then Practical Byzantine Fault Tolerance is the best solution. It has a very high throughput that is it achieves thousands of transactions with simple state machine replication. It is this main reason, that is has been adopted by Hyper-ledger family [41, 55]. Hyperledger Iroha [54], a blockchain platform for IoT devices - specialized designed to efficiently work with IoT devices. If devices are to be deployed in open settings, then Algorand offers the solution. This is computationally expensive than the PBFT, but true decentralization and transaction scalability comes at cost of that.

Chapter 8

Open Challenges and Proposed Architectural Solution

8.1 Open Challenges in Blockchain

Though blockchain has emerged as the revolutionary technology that has impacted all major businesses ranging from finance, data storage, inventory tracking, medical to internet of things and many more for its unique characteristics. But it fails to cope with the massive business solutions. Enhancing blockchain's transaction and network scalability to match the scale of today's systems, while maintaining the trust in an open public without intermediary is an active area of research.

8.1.1 Enhancing Network Scalability

Network scalability is usually high for blockchains designed to operate in open settings (permissionless environments). PoX and hybrid blockchains mostly fall in this domain. However, mostly vote based blockchain are designed to operate in closed settings (permissioned environments) usually have a limited network scalability. Once, number of nodes exceed a certain limit due to message overhead congestion occurs on network, thus limiting the network scalability of these chains. Certain sharding solutions have been proposed for the consensus in permissionless settings, but none has been for the consensus in permissioned environment [33].

8.1.2 Reduction of Message Complexity for Classical BFT Protocols

Classical BFT consensus protocols scale well in close settings up to a certain limit and beyond that point their performance degrades severely due to number of message exchanged during different stages to reach consensus. However, recently researchers have started exploring techniques where message complexity can be reduced from $O(N^2)$ to something less like $O(N)$. One such technique is used in ByzCoin 6.3 is collective signing [67].

8.1.3 Enhancing Transaction Scalability

In blockchain usually a large number of devices have to talk to each other. This demands a high throughput system. But in existing blockchain systems, once transaction speed is enhanced, it adversely effects network scalability and security of the system. Thus, mostly a trade off is struck between these two. However, efforts are being made to achieve both at the same time. Proof of Stake 4.3 [57], sharding and side chains are active research areas of this domain.

8.1.4 Storage Limitations

In existing blockchain protocols, all the nodes participating in consensus, store entire data set with them. However, with passage of time blockchain size grows and hence the storage requirement also increases. Currently Bitcoin has grown to 226,596 Mega Bytes [6], and is still growing. Such massive growth of data may create problem for widley used blockchains. Thus it is need that some efficient pruning strategies be devised to handle this growing demand of storage.

8.1.5 Reducing Computational Requirements

Most of the existing implementations of blockchain protocols like Bitcoin and Ethereum heavily rely on sophisticated cryptographic computations to achieve security. However, not all devices are that powerful especially the IoT devices. Thus, energy efficient consensus protocols with similar security assurities are to be designed.

8.2 Proposed Architectural Solutions

People have explored various approaches to overcome the short comings of blockchain and tried to close few of the gaps mentioned in aforementioned paragraphs. One very simple approach is to adjust the parameters of existing protocols like its block size, confirmation time for transactions etc. Such an approach will achieve specific aims in specific environment under specific requirements but will jeopardise the security of the overall system and may fail completely in other environments. For example study [112] demonstrates that maximum throughput of the Bitcoin network can be achieved at 4 MB block size. However, such modifications will create congestions in low bandwidth networks. Similarly, once there parameters are adjusted these have direct impact on the security of the reference blockchain.

Another disadvantage of such approaches is that these make the updated blockchain incompatible with the other implementations of the same blockchain. Thus people have worked out other solutions like side-chains, off-chains and multi-chains, those aim at achieving same security assumption as of the main blockchains today while improve upon its throughput. Contrary to other solutions these stay compatible with the existing implementations (referred as 'main' chains) and sometimes referred as *Layer 2 Solutions* [74].

8.2.1 Side Chains

Side Chain provide a solution where a different blockchain other than the main blockchain which is working at higher throughput is connected to the some main chain like Bitcoin. Side chains usually have very high throughput but low security and mostly have properties totally different from the main chain. Side chain protocol enables transaction between the main chain and the side chain. Side chains exist independently and usually have minimal interaction with the main chain. However, side-Chain is the custodian of the same asset as the main chain [86] i.e those do share the common asset. Those are most suited for the systems where most of the operations do involve the participant of the main chain i.e most of the transactional data stored on the local side chain is complete in itself. For example if a side chain is handling a micro payment system, it is not necessary that every transaction is recorded on the main chain especially if the transactions are recurring. Involved parties carry out their recurring transactions on the local chain and their final balances are stored onto the main chain. This way both high throughput for transactions and security is achieved. Different mechanisms are used to achieve side chains functionality. These include Relays, Hash locks and Bridges or gateways [23].

8.2.2 Off Chains

Off chain protocols are usually devised to free the main chains from burden of heavy computations. Aim here is to record heavy transactions offline. Once those transactions have been finalized, synchronize the finalised results with the main chain. Thus use the main chain for only book keeping purpose. One such protocol is Plasma [87]. It plasma, child chains are linked to main Ehtereum Chain through smart contracts. Child chains can perform complex smart contracts or any other tasks without interfering with the main chain. As there can be multiple off chains, those can perform desired operations and write the final outcomes to the

main chain as an authenticated proof. Off chain solutions have been encouraged specifically where large amounts of data has to be stored on the chain like a blockchain based cloud storage solution.

8.2.3 Sharding

Sharding has been historically used in databases to make them more efficient and robust by horizontally dividing the database. In case of Blockchain, sharding has two notions:

1. Sharding of the data stored with mining nodes. Without shards every node keeps an entire copy of the transactions history with itself. Sharding will allow nodes to keep only the data that is relevant to them [37].

2. Sharding of the nodes themselves. Divide nodes in shards and then use some vote based mechanism to quickly reach consensus.

Chapter 9

Conclusion

This research provides an overview of blockchain functionality independent of any specific implementation. It compares nine consensus protocols namely Nakamoto consensus as in Bitcoin, Nakamoto consensus as in Bitcoin-NG, Proof of Stake as in Ouroboros family, Practical Byzantine Fault Tolerance state machine replication, BFT-Smart, BA* as in Algorand, Elastico, Omni Ledger and ByzCoin for various parameters related to security like adversary tolerance and consensus finality, and performance like transaction throughput and latency along with a thorough introduction and functioning of the compared protocols. It also hints upon suggested consensus protocols for IoT devices and open challenges in consensus domain related to mainly transaction scalability and security. Comparison concludes that there is no winner in the race for the best consensus protocol. It is all about trade-offs, either on security or performance.

Parametrized data for various protocols along with their functioning details will act as a ready reckoner for the future studies and developers.

BIBLIOGRAPHY

[1] Aeternity blockchain. https://blog.aeternity.com/introduction-to-%C3%A6ternitys-bitcoin-ng-implementation-331d0a1393b4.

[2] Ieee 802.15.4. wikipedia.org.

[3] Merkle tree. Wikipedia.

[4] Ouroboros hydra - Developers - Cardano Forum.

[5] PSA: F2Pool is mining INVALID blocks : Bitcoin. reddit.com, 2015.

[6] Bitcoin blockchain size 2010-2019 — Statista, 2019.

[7] Aeternity. Aeternity blockchain code.

[8] Aggelos Kiayias. IOHK presents at Oxford University: Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol - YouTube, 2019.

[9] Elli Androulaki, Ghassan O. Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating user privacy in Bitcoin. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7859 LNCS:34–51, 2013.

[10] Andreas M. Antonopoulos. *Mastering Bitcoin*. 2010. wallets forks bitcoin consensus + conditions change for bitcoin core.

[11] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. Hijacking bitcoin: Large-scale network attacks on cryptocurrencies. *arXiv preprint arXiv:1605.07524*, 2016.

[12] Algirdas Avižienis, Jean Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[13] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. Consensus in the age of blockchains. *arXiv preprint arXiv:1711.03936*, 2017.

[14] Imran Bashir. *Mastering Blockchain, Second Edition*. Packt Publishing, second edition, 2018.

[15] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work. In *International Conference on Financial Cryptography and Data Security*, pages 142–157. Springer, 2016.

[16] Pascal Berrang. Survey of Consensus Protocols and Scalability Solutions. pages 1–14, 2015.

[17] Stefan Beyer. Enterprise blockchain platforms - a comparison. *Web*.

[18] Bitnodes. Network Snapshot - Bitnodes. Https://bitnodes.earn.com/, April 2019.

[19] Blockchain.com. Market Capitalization - Blockchain, 2019.

[20] Jonas Bostoen. A comparison of blockchain platforms - competitors. web, 2018.

[21] Brenn Hill, Samanyu Chopra, and Paul Valencourt. *Blockchain Quick Reference*. Packt Publishing, first edition, 2018.

[22] Richard Gendal Brown, James Carlyle, Ian Grigg, and Mike Hearn. Corda: an introduction. *R3 CEV, August*, 1:15, 2016.

[23] Vitalik Buterin. Chain interoperability. *R3 Research Paper*, 2016.

[24] Christian Cachin. Architecture of the hyperledger blockchain fabric. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, volume 310, 2016.

[25] Christian Cachin and Marko Vukolić. Blockchain consensus protocols in the wild. *arXiv preprint arXiv:1707.01873*, 2017.

[26] Cardano. Cardano. web.

[27] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance. *OSDI '99: Proceedings of the Third Symposium on Operating Systems Design and Implementation SE - OSDI '99*, (February):173–186, 1999.

[28] Miguel Castro and Microsoft Research. Practical Byzantine Fault Tolerance - YouTube, 2016.

[29] Charles. EMURGO Cardano Meetup ft. Charles Hoskinson - YouTube.

[30] Allen Clement, Manos Kapritsos, Sangmin Lee, Yang Wang, Lorenzo Alvisi, Mike Dahlin, and Taylor Riche. Upright cluster services. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 277–290. ACM, 2009.

[31] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jeffrey John Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, et al. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS)*, 31(3):8, 2013.

[32] Flaviu Cristian, Houtan Aghili, Raymond Strong, and Danny Dolev. *Atomic broadcast: From simple message diffusion to Byzantine agreement*. International Business Machines Incorporated, Thomas J. Watson Research Center, 1986.

[33] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. On Scaling Decentralized Blockchains. *International Conference on Financial Cryptography and Data Security*, pages 106–125, 2016.

[34] dedis. Byzcoin. github.com, 2019.

[35] Xavier Défago, André Schiper, and Péter Urbán. Total order broadcast and multicast algorithms. *ACM Computing Surveys*, 36(4):372–421, 2005.

[36] Daniel Drescher. *BLOCKCHAIN BASICS A Non-Technical Introduction in 25 Steps*. Apress Business, 2017.

[37] edChain. Blockchain faq #3: What is sharding in the blockchain? medium.com, May 2018.

[38] Joerg Evermann and Henry Kim. Workflow Management on the Blockchain — Implications and Recommendations. pages 1–15, 2019.

[39] Ittay Eyal, Adem Efe Gencer, Emin Gun Sirer, and Robbert van Renesse. Bitcoin-NG: A Scalable Blockchain Protocol. 2015.

[40] Reddit Forum. Maximum block gas limit. https://www.reddit.com/r/ethereum/comments/87qt04/.

[41] Linux Foundation. Hyperledger, 2018.

[42] Alexis Gauba. Finality in blockchain consensus. *https://medium.com/*, 2018.

[43] Alexis Gauba. Finality in Blockchain Consensus - Mechanism Labs - Medium, 2018.

[44] Gazi Peter. IOHK — Dr. Peter Gaži, presenting Ouroboros at MIT. - YouTube.

[45] Arthur Gervais, Hubert Ritzdorf, Ghassan O Karame, and Srdjan Capkun. Tampering with the delivery of blocks and transactions in bitcoin. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 692–705. ACM, 2015.

[46] Elli Audroulaki Ghassan Karame. *Security of Transactions in Bitcoin*, chapter 4, pages 66–90. Artech House, 1st edition, 2018.

[47] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68. ACM, 2017.

[48] Seth Gilbert, Calvin Newport, and Chaodong Zheng. Who are you? secure identities in ad hoc networks. In *International Symposium on Distributed Computing*, pages 227–242. Springer, 2014.

[49] Andrei Grigorean. Latency and finality in different cryptocrurrencies. hackernoon.com, May 2018.

[50] Jan Hackfeld. A lightweight BFT consensus protocol for blockchains. pages 1–21, 2019. liveness and safety.

[51] Yunhua He, Hong Li, Xiuzhen Cheng, Yan Liu, Chao Yang, and Limin Sun. A Blockchain Based Truthful Incentive Mechanism for Distributed P2P Applications. *IEEE Access*, 6(c):27324–27335, 2018.

[52] Subhrojit Nag Hemant Sachdeva. Techwatch report - blockchain frameworks. report, talentica, 2018.

[53] John Hinkle. The fastest cryptocurrency transaction speeds for 2018. medium.com, October 2018.

[54] Hyperledger. Hyperledger iroha, 2018.

[55] HyperledgerDocs. Introduction — hyperledger-fabricdocs master documentation. https://hyperledger-fabric.readthedocs.io/en/release-1.4/whatis.html, 2018.

[56] IIT Kharagpur and Sandip Chakarborty. Lecture 52: Research Aspects – V (Algorand - I) - YouTube, 2018.

[57] Akshita Jain, Sherif Arora, Yashashwita Shukla, T B Patil, and S T Sawant-patil. Proof of stake with Casper the friendly finality gadget protocol for fair validation consensus in Ethereum. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 3(3):291–298, 2018.

[58] Markus Jakobsson and Ari Juels. Proofs of work and bread pudding protocols. In *Secure Information Networks*, pages 258–272. Springer, 1999. proof of work concept idea.

[59] V. Buterin James Ray. Ethereum - a next-generation smart contract and decentralized application platform. Online, 2018.

[60] Jimi S. Blockchain terminology; the 35 most commonly used blockchain terms explained., 2017. Proof of works is the Most widely used consensus.

[61] Jordan Tuwiner. 10 Best and Biggest Bitcoin Mining Pools 2019 (Comparison), 2019.

[62] Ghassan Karame and Elli Androulaki. *Bitcoin and blockchain security*. 2016.

[63] Aggelos Kiayias and Giorgos Panagiotakos. Speed-security tradeoffs in blockchain protocols. *IACR Cryptology ePrint Archive*, 2015:1019, 2015.

[64] S King and S Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. peercoin whitepaper, 2012.

[65] S King and S Nadal. Peercoin–secure & sustainable cryptocoin. *Aug-2012 [Online]. Available: https://peercoin. net/whitepaper ()*, 2018.

[66] Tommy Koens and Erik Poll. What blockchain alternative do you need? In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 113–129. Springer, 2018.

[67] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 279–296, 2016.

[68] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. *Proceedings - IEEE Symposium on Security and Privacy*, 2018-May:583–598, 2018.

[69] Jae Kwon. Tendermint: Consensus without mining. *Draft v. 0.6, fall*, 1:11, 2014.

[70] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[71] Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. Reconfiguring a state machine. *SIGACT News*, 41(1):63–73, 2010.

[72] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[73] Linuxfoundation. GitHub - bitcoin/bips: Bitcoin Improvement Proposals.

[74] Peter Lipovyanov. *Blockchain for business 2019: a user-friendly introduction to blockchain technology and its business applications*. Packt Publishing Ltd, 2019.

[75] Pablo Lorenceau, Prof Bryan Ford, and Epfl Dedis. An Implementation of Omniledger. *GitHub*, (June), 2018.

[76] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A Secure Sharding Protocol For Open Blockchains. pages 17–30, 2016.

[77] Nancy A Lynch. *Distributed algorithms*. Elsevier, 1996.

[78] Yuval Marcus, Ethan Heilman, and Sharon Goldberg. Low-resource eclipse attacks on ethereum's peer-to-peer network. *IACR Cryptology ePrint Archive*, 2018:236, 2018.

[79] Mainak Chatterjee Mehrdad Salimitari. An overview of blockchain and consensus protocols for iot networks. 2018.

[80] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 120–130. IEEE, 1999.

[81] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. Permacoin: Repurposing bitcoin work for data preservation. *Proceedings - IEEE Symposium on Security and Privacy*, pages 475–490, 2014.

[82] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[83] nxtcrypto.org. Next Generation of Cryptocurrency • NxtCoin. web page.

[84] Diego Ongaro and John K Ousterhout. In search of an understandable consensus algorithm. In *USENIX Annual Technical Conference*, pages 305–319, 2014.

[85] Santeri Paavolainen and Pekka Nikander. Security and privacy challenges and potential solutions for DLT based IoT systems. *2018 Global Internet of Things Summit, GIoTS 2018*, (June):1–6, 2018.

[86] Babu Pillai, Kamanashis Biswas, and Vallipuram Muthukkumarasamy. *Blockchain Interoperable Digital Objects*, pages 80–94. 06 2019.

[87] Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts. *White paper*, pages 1–47, 2017.

[88] Narayan Prusty. *Blockchain for Enterprise: Build scalable blockchain applications with privacy, interoperability, and permissioned features*. Packt Publishing Ltd, 2018.

[89] Alexander Renkl Erstgutachter, Hc Günter Müller Zweitgutachter, and Dieter K Tscheulin. Inaugural-Dissertation Transparency through Decentralized Consensus: The Bitcoin Blockchain and Beyond. 2016.

[90] Ana Reyna, Cristian Martín, Jaime Chen, Enrique Soler, and Manuel Díaz. On blockchain and its integration with iot. challenges and opportunities. *Future Generation Computer Systems*, 88:173–190, 2018.

[91] Tim Ruffing, Pedro Moreno-sanchez, and Aniket Kate. CoinShuffle: Practical Decentralized Coin Mixing for Bitcoin – Bookmetrix Analysis. *Computer Security - ESORICS*, 8713:1–15, 2014.

[92] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 515–532. Springer, 2016.

[93] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9603 LNCS:515–532, 2017.

[94] Richard D. Schlichting and Fred B. Schneider. An approach to designing fault-tolerant computing systems. *ACM Trans. Comput. Syst.*, 1:222–238, 1981.

[95] Fred B Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990.

[96] Ronald Sonntag. Blockchain scalability: When, where, how? https://blockgeeks.com/guides/blockchain-scalability/, 2018.

[97] João Sousa, Eduardo Alchieri, and Alysson Bessani. State machine replication for the masses with bft-smart. 2013.

[98] João Sousa and Alysson Bessani. Separating the wheat from the chaff: An empirical design for geo-replicated state machines. In *2015 IEEE 34th Symposium on Reliable Distributed Systems (SRDS)*, pages 146–155. IEEE, 2015.

[99] Nicholas Stifter, Aljosha Judmayer, Philipp Schindler, Alexei Zamyatin, and Edgar Weippl. Agreement with Satoshi – On the Formalization of Nakamoto Consensus. pages 1–17.

[100] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 444–460. Ieee, 2017.

[101] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities" honest or bust" with decentralized witness cosigning. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 526–545. Ieee, 2016.

[102] Shunsai Takahashi. Proof of work - A better Consensus. medium.com, 2019. 25

[103] Luca Favatella Tobias Lindahl, Micha. protocol/bitcoin-ng.md at master · aeternity/protocol · GitHub. https://github.com/aeternity/protocol/blob/master/consensus/bitcoin-ng.md, 2019.

[104] Vitalik Buterin. Bitcoin News: Bitcoin Network Shaken by Blockchain Fork, 2013.

[105] Vitalik Buterin. On Settlement Finality, 2016.

[106] Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9591:112–125, 2016.

[107] Wenbo Wang, Dinh Thai Hoang, Peizhao Hu, Zehui Xiong, Dusit Niyato, Ping Wang, and Yonggang Wen. A Survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks. 2018.

[108] Wikipedia. Gossip protocol. wikipedia.org. gossip protocol.

[109] Wikipedia. Smart contract. https://en.wikipedia.org/wiki/Smart_contract.

[110] wikipedia. Consensus decision making. https://en.wikipedia.org/wiki/Consensus_decision-making, 2017.

[111] GAVIN(ETHEREUM) WOOD. Ethereum: a Secure Decentralised Generalised Transaction Ledger. *Ethereum Project Yellow Paper*, 32(10):1365–1367, 2018.

[112] Karl Wüst, Hubert Ritzdorf, Ghassan O. Karame, Vasileios Glykantzis, Srdjan Capkun, and Arthur Gervais. On the Security and Performance of Proof of Work Blockchains. pages 3–16, 2016.

[113] Yang Xiao, Ning Zhang, Wenjing Lou, and Y Thomas Hou. A survey of distributed consensus protocols for blockchain networks. *arXiv preprint arXiv:1904.04098*, 2019.

[114] Xiwei Xu, Ingo Weber, and Mark Staples. *Architecture for Blockchain Applications*. Springer, 2019.

[115] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 931–948. ACM, 2018.