4-9-2020

Scaling Private Collaborated Consortium Blockchains Using State Machine Replication Over Random Graphs

Parth Shukla
University of Windsor

Scaling Private Collaborated Consortium Blockchains Using State Machine Replication Over Random Graphs

By

Parth Anand Shukla

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science
at the University of Windsor

Windsor, Ontario, Canada

2020

Scaling Private Collaborated Consortium Blockchains Using State Machine Replication Over Random Graphs

by

Parth Anand Shukla

APPROVED BY:

M. Kianieff
Faculty of Law

P. Moradian Zadeh
School of Computer Science

S. Samet, Advisor
School of Computer Science

February 13, 2020

DECLARATION OF CO-AUTHORSHIP / PREVIOUS PUBLICATION

I hereby declare that this thesis this includes all the details and experiments and system design which was developed under the supervision of my advisor, Dr. Saeed Samet. All the contributions, experimental ideas, design, system testing, analysis and interpretation were performed by the author and the contribution of my supervisor, Dr.Saeed Samet was majority through proofreading of thesis and published papers.

I am aware of the University of Windsor Senate Policy on Authorship and I certify that I have properly acknowledged the contribution of other researchers to my thesis, and have obtained written permission from each of the co-authors to include the above materials in my thesis. I certify that, with the above qualification, this thesis, and the research to which it refers, is product if my own work.

This thesis includes one original paper that has been previously submitted in peer reviewed conference as follows:

Thesis Chapter	**Publication Title**	**Status**	**Citation**
Chapter 3	Systematizaton of Knowledge on Scalability Aspect of Blockchain Systems	Accepted	P. A. Shukla and S. Samet, Systematization of Knowledge on Scalability Aspect of Blockchain Systems, in Advances in Information and Communication, Cham, 2020, pp. 130138.

I certify that I have obtained a written permission from the copyright owners to include the above published material in my thesis. I certify that the above material describes the work completed during my registration as a graduate student at the

University of Windsor.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyones copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

Blockchain technology has redefined the way the software industry's core mechanisms operate. With recent generations of improvement observed in blockchain, the industry is surging ahead towards replacing the existing computing paradigms with consortium blockchain-enabled solutions. For this, there is much research observed which aims to make blockchain technologys performance at par with existing systems. Most of the research involves the optimization of the consensus algorithms that govern the system. One of the major aspects of upcoming iterations in blockchain technology is making individual consortium blockchains collaborate with other consortium blockchains to validate operations on a common set of data shared among the systems. The traditional approach involves requiring all the organizations to run the consensus and validate the change. This approach is computationally expensive and reduces the modularity of the system. Also, the optimized consensus algorithms have their specific requirements and assumptions which if extended to all the organizations leads to a cluttered system with high magnitudes of dependencies.

This thesis proposes an architecture that leverages the use of state machine replication extended to all the nodes of different organizations with seamless updates over a random graph network without involving all the nodes participating in the consensus. This also enables organizations to run their respective consensus algorithms depending on their requirements. This approach guarantees the finality of consistent data updates with reduced computations with high magnitudes of scalability and flexibility.

DEDICATION

Dedicated to my mother ***Swati Anand Shukla*** *and my grandmother* ***Gita*** *for their inundated enthusiasm and love for me.*

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to **Dr. Saeed Samet** for his continuous support and inputs through out my work. I could not have imagined better mentor for my study. I would also like to thank **Dr. Pooya Moradian Zadeh** and **Dr. Muharem Kianieff** for their inputs that helped me in refining and improving my work and helping me in guiding my work in proper direction and leading my thesis towards gracious completion.

Also, I would like to thank my batch mates **Dipesh Patel**, **Lokesh Gupta**, **Jayanth Kulkarni** and **Saurav Agarwal**, for keeping me encouraged, without whom I could not have imagine my journey at the University. Special thanks to my best friend **Aaryaman Anerao** for always being there.

TABLE OF CONTENTS

DECLARATION OF CO-AUTHORSHIP / PREVIOUS PUBLICATION **III**

ABSTRACT **V**

DEDICATION **VI**

ACKNOWLEDGEMENTS **VII**

LIST OF TABLES **X**

LIST OF FIGURES **XI**

1 Introduction **1**
1.1 State of Blockchain Technology 1
1.2 Motivation Behind The Work 2
1.3 Problem Statement & Outlined Solution 5
1.4 Structure of Thesis 5

2 Overview of Related Technologies **7**
2.1 Blockchain Systems 7
2.2 Consensus Algorithm 9
2.2.1 Proof of Work 9
2.2.2 Practical Byzantine Fault Tolerance 10
2.2.2.1 Byzantine General's Problem 11
2.2.2.2 Working of PBFT 12
2.3 Random Graph Theory 13
2.3.1 Understanding Graph Theory Basics 13
2.3.2 Random Graph 14
2.4 State Machine Replication 17

3 Related Works **19**
3.1 Approaches to Scale Blockchain System 19
3.1.1 Quorum Formation 19
3.1.2 Communication Topology 21
3.1.3 Cryptographic Techniques 22
3.1.4 Trusted Hardware Components and Trusted Execution Environments 23
3.1.5 Off-Chain Paradigm 24
3.1.5.1 Off-Chain Storage 24
3.1.5.2 Off-Chain Computing 25
3.2 Gossiping in Distributed Systems 26

4 Methodology **28**
4.1 Node State Parameters 29
4.2 Informed Gossip 30
4.3 Proposed Solution 33

5 Experimental Results & Analysis **35**
5.1 Complexity Analysis 35
5.2 Empirical Analysis 36
5.2.1 Scalability Aspect of Model - Number of Connections vs Number of Nodes 36
5.2.2 Effects of itr and p on Model 37
5.2.3 Degree Distribution and Connectivity of Graph 42
5.3 Implementation 44
5.3.1 Platform Technology 44
5.3.2 Blockchain Specifications & System Design 44
5.3.3 Results 46

6 Conclusion & Future Work **50**
6.1 Conclusion 50
6.2 Future Work 51

REFERENCES **52**

VITA AUCTORIS **57**

LIST OF TABLES

4.2.1 Logical View of Node Index 30

5.2.1 *itr* experiment details 41
5.2.2 Overall Results of Variation Test 41
5.3.1 System Configuration - Test Bed 46
5.3.2 Node Configuration - Container Specification 46
5.3.3 Transaction Throughput Results (tps) 48
5.3.4 Transaction Latency Results (ms) 48
5.3.5 Computational Complexity of Methods 49

LIST OF FIGURES

1.2.1 IODM Decision Tree 3

2.1.1 Contents of a Block 8
2.2.1 Byzantine General's Problem Setup 11
2.2.2 PBFT Communication Flow 12
2.3.1 A Sample Graph 13
2.3.2 $G(N, L)$: $G(5, 4)$ graph 16

3.1.1 Approach to Scalable BFT Design [4] 20
3.1.2 FastBFT Communication Flow with $b = 2$ 21
3.1.3 LinBFT Communication Flow 22
3.2.1 General Structure of Gossiping Algorithm [21] 27

4.2.1 Informed Gossip Example 30

5.1.1 Logical View of Node Dissemination 35
5.2.1 Growth of Edges with increasing number of nodes 36
5.2.2 $G(100, 0.5)$ with $itr = ln(N)$ 37
5.2.3 $G(100, 0.6)$ with $itr = ln(N)$ 38
5.2.4 $G(100, 0.7)$ with $itr = ln(N)$ 38
5.2.5 $G(100, 0.8)$ with $itr = ln(N)$ 38
5.2.6 $G(100, 0.5)$ with $itr = ln(N)$ 39
5.2.7 $G(100, 0.5)$ with $itr = ln(N) + 1$ 40
5.2.8 $G(100, 0.5)$ with $itr = ln(N) + 2$ 40
5.2.9 $G(100, 0.6)$ over 100 experiments 41
5.2.10 Degree Distribution - In Degree 42
5.2.11 Degree Distribution - Out Degree 43
5.2.12 Degree Distribution - Total Degree 43
5.2.13 Degree Distribution - Generated Graph 43
5.3.1 User Account 44

5.3.2 Blockchain Ledger 45

CHAPTER 1

Introduction

1.1 State of Blockchain Technology

In the present years, Blockchain technology has gained strong attention in the tech community and other industries where it seems to be a viable refinement, an addition or a replacement of conventional computing paradigm in the technology stack [5]. The substantial reason behind this charisma is the three quintessential properties that it adheres. First, an *immutable ledger* formed by hashing the transaction blocks with each other forming a chain which is also responsible for the second property i.e., *transparent audit trail* which can help organizations to trace the operations and state changes inside the blockchain and third, *high availability of data* which is attained by replicating the data across every peer in the network making system tolerant to the single point of failure.

Blockchain is a completely decentralized system that is governed by a set of instructions that makes sure that all the nodes in the system maintain a common state. These instructions are preached in the form of consensus algorithms. A decentralized system without a common consensus will disintegrate, regardless of the participants in the system trust each other particularly or do not trust at all. Solid governance is the key to a working decentralized system that is fulfilled by consensus algorithms. There is no specific standard for consensus algorithms. It essentially depends on the use-case scenario and requirements of the system.

Blockchain technology has seen three major iterations until now. **First generation** was originated with *Bitcoin* [30] which essentially laid foundations of a fully

functional decentralized & trust-less technology. Bitcoin was an implementation of blockchain technology which was based on the concept of cryptocurrency. First-generation brought realizations regarding extending the technology to a higher spectrum of use-cases, which lead to the **second generation** of blockchains which involved *Smart Contracts & Tokenization* which was brought in by Ethereum [10]. The second-generation proposed a view to tokenize any physical and digital assets without enforcing platform ownership. Finally, **third generation** is essentially industry-ready design [1] of blockchains which are having a performance at par with present systems and can be work as a valuable addition to the technology stack of the organizations. The third generation is the future and the present industry stands on the transition phase, where the research is surging ahead making blockchain viable for industry. Currently, blockchains have not been widely accepted despite wide spectrum of use-cases is because of the major problem that this technology faces, i.e., performance scaling which is resultant of various aspects starting from the computational complexity of the consensus algorithms, fault tolerance of system in terms of Byzantine faults and expensive storage needs as the continuously growing ledger with respective data entities needs to be replicated over entire network nodes [39].

1.2 Motivation Behind The Work

With the technology moving towards the third generation, trying to cover the advance use-cases like digital identity management, supply chain management, audit trails, large scale logging, automated governance, IoT, banking, health care systems, etc. Blockchain is allowing people to secure digital relationships with regards to their assets by harnessing the power of cryptography in the hands of people. To make the blockchain use-case at par with the existing systems, scaling of blockchains has been widely addressed as the most important research area in this field.

One can observe that in toady's scenario multiple organizations tend to share data and the same share of data is updated on both the ends and updates are collectively shared among the organizations. This transformation and sharing of data are man-

aged by a system design that is called Inter-Organization Data Management (IODM). The decision tree shown in Figure 1.2.1 highlights the usability and adaptability understanding of IODMs in the industry [37].

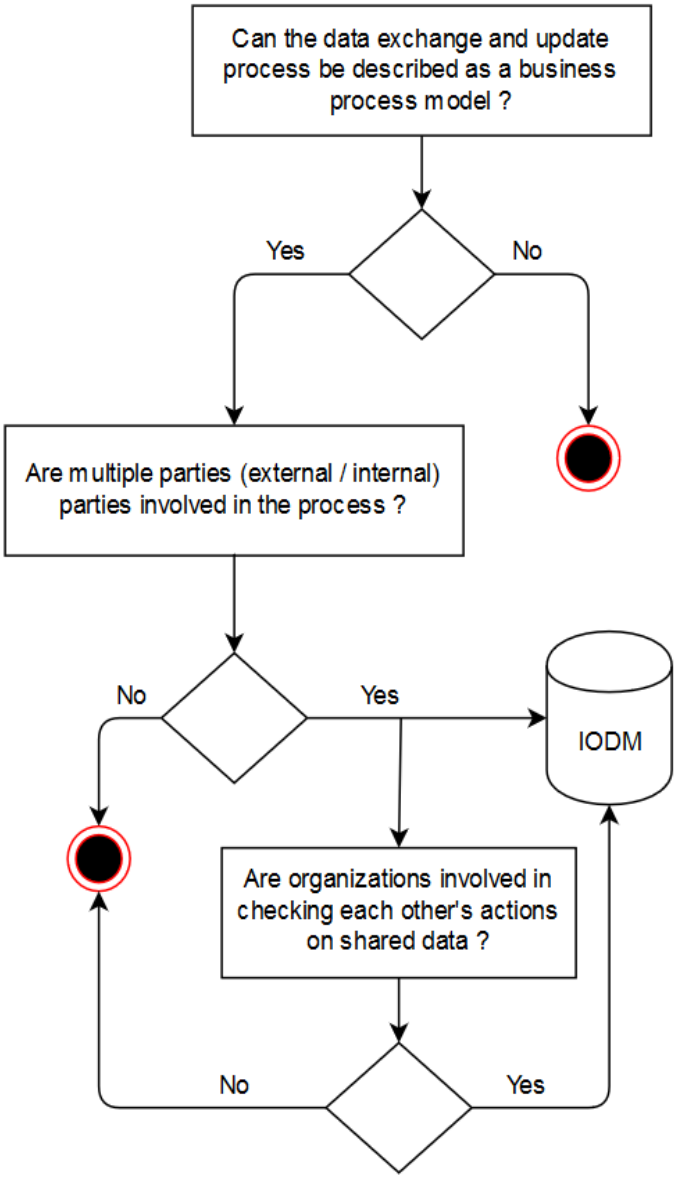

Fig. 1.2.1: IODM Decision Tree

Figure 1.2.1 describes the road-map that leads the industry to consider IODM systems as part of their day to day operations. The first criterion that is taken into consideration is that, if there is any kind of data shared or retrieval of data from other sources can be part of the computations. If yes, then it leads to second conditions that

check if the fore stated condition is valid given there will be multiple parties involved in the process. If yes, then it leads to an organization or group of organizations consider the IODM system. Furthermore, to get decisions more convincing, the second condition is branched into another question, which asserts that if multiple parties are authorized to audit the operations performed on the shared data, in that case, IODM is a sure thing.

IODMs have been used in many industries, such as supply-chain management, fin-tech organizations, Health Record systems, loggers, and other Data Governance systems. Current solutions that industries adhere to are centralized in nature because the current state of computing systems is not decentralized. With these companies trying to surge towards blockchain as their core computing stack, using centralized IODMs is not possible.

Data is managed and operations are validated with help of consensus algorithms on the blockchain which is computationally complex and expensive in some cases. Consensus algorithms are not a generic entity. It is not a standard that can be the same for all the organizations because consensus algorithms are designed based on the organization's requirement. So considering a supply chain management group where 3 organizations have 100 nodes each as part of their blockchain, each have their own consensus mechanisms, in order to make sure that all 3 organizations stay in sync with respect to the operations performed is only possible when all 300 nodes take part into the consensus process and validate the change. Assuming, all 3 organizations are aligned to change, even a slight change to the consensus algorithm will lead to a feature update or patch in all the nodes. This makes design monolithic and cohesive in nature which is not a good property for decentralized and distributed systems. Also, this questions scalability of the system because adding more organizations to the group will lead to higher degrees of cohesiveness. This thesis tries to solve this problem for blockchain which is a big thorn on the pathway blocking the surge towards fully functional and scalable third-generation blockchain technology.

The current state of consensus algorithms is continuous refinement by optimizing various parts and tweaking parameters leading to faster executions, but the collab-

orative blockchains have never been considered as part of future use-case scenarios. Consensus algorithms in the current state mostly consider the local state of the system than the global state which is extensive and quintessential to make centralized technologies decentralized.

1.3 Problem Statement & Outlined Solution

As discussed in the section above, we can formalize the problem statement as follows: *Given a set of consortium blockchains, sharing a common set of data, validating the transactions using their own specific consensus algorithms, device architecture which helps the individual blockchains collaborate over a common medium helping achieving a common state of data without making all the blockchains participate into consensus.*

We tackle the problem with points stated below which we discuss comprehensively in subsequent chapters of thesis.

- Keeping consensus execution to the validating organization's blockchain.
- Extending the logic to interpret finality of consensus (f^*) to rest of the network which will serve as base for a consistent state using **State Machine Replication**.
- Dissemination of the message from validating blockchain to rest of the blockchain in a decentralized fashion is done in an optimized communication topology that is resultant of **informed gossip** over the **random graph**.

1.4 Structure of Thesis

The subsequent chapters of the thesis are organized as follows. In **Chapter 2**, we present the readers with succinct knowledge which lays a technical foundation for the rest of the thesis. As for **Chapter 3**, we present the Literature Survey of related work done in the area. **Chapter 4** explains the implemented Methodology to solve the problem. **Chapter 5** includes the experiment results and analysis of the proposed

solution and comparison with other methods. And lastly, In **Chapter 6**, we conclude the thesis and address the work that can be possibly done future to refine the proposed solution.

CHAPTER 2

Overview of Related Technologies

2.1 Blockchain Systems

Blockchain technology is a **append-only** data-structure that stores a continuously growing list of operations performed on the data which is replicated over a set of nodes governing the system. Blockchains are completely **decentralized** with no intermediaries governing the state of the system. Data in blockchains is replicated over the entire set of nodes governing the system, eradicating articulation points in the system making system immune to a single point of failure. The changes stored in the system are cryptographically secure such that, any manipulation will lead to a completely disrupted chain of records making system immutable, promising **integrity** and **non-repudiability** of data.

The fundamental unit of a blockchain is called a block. Every block is associated with a transaction. A transaction is any kind of operation that updates data managed by the blockchain system. Multiple transactions based on their occurrences and processing time are chained in a linear pattern in a cryptographically secured manner and hence the term blockchain. Figure 2.1.1 shows the content of the block in the blockchain.

block_id is a unique identifier for a block in the blockchain. *merkle_root* is the resultant root of the Merkle tree [35] formed by the subsequent hashing of the transactions from the *set_of_transactions*. *time_stamp* represents the block creation time. *meta_data* field is the general information about the blockchain like what kind of standards it follows for consensus and cryptography algorithm standards etc.

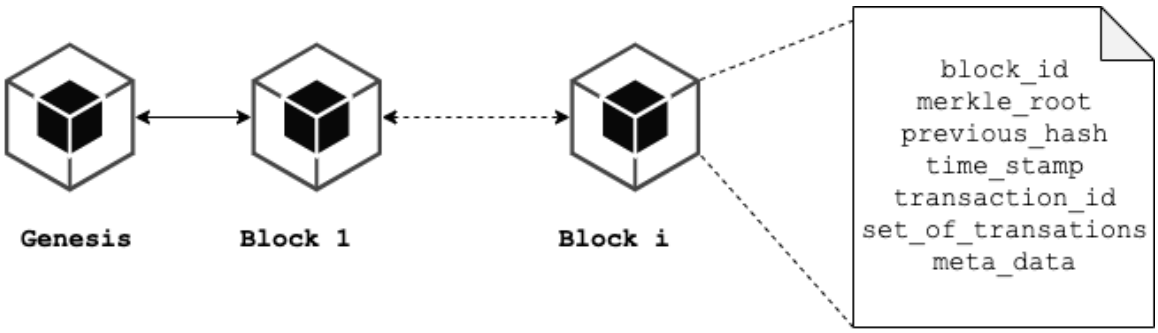

Fig. 2.1.1: Contents of a Block

Based on how nodes participate in the system, there are essentially two types of blockchains i.e. public blockchain and private/consortium blockchain. The system design and overall considerations taken up in these two blockchains is the biggest differentiating factor among them.

1. Public Blockchain

   - Any node can essentially participate into in the consensus process, hence the word public blockchain
   - A node may join and may leave at any time. Consensus protocols and other validation processes should align with the security requirements and volatility of the size of the validation group.
   - Because of security considerations and uncertainty of system, it is not preferred in industries.

2. Consortium / Private Blockchain

   - Private blockchains are governed by an organization or a group of organizations. Nodes specific to the organizations can only be part of the blockchain.
   - There are no strong security considerations as the system is not open to public access and the volatility of the validation group is not severe as the node's state is under the organization's control.
   - A widely accepted variant of blockchain in industry.

2.2 Consensus Algorithm

In a distributed system, the nodes participating in the system are part of a common set of a protocol that is to be followed to ensure that the system works in the desired manner. In a decentralized and self-governed distributed system like blockchain, a sense of orchestration between nodes is essential to maintain a consistent and valid state. For this, **consensus algorithm** is used in the blockchains.

Regardless of the participants in the system trust each other particularly or do not trust at all, a decentralized system without a common consensus will disintegrate. Solid governance is key to a working decentralized system. The consensus algorithm makes blockchain operations orchestrate in a defined set of rules in all the nodes.

The **state** of a blockchain is defined by the present values and blocks that the distributed ledger holds. A consistent state is nothing but blocks and values generated by the proper execution of the consensus algorithm. Therefore, we can say that the consensus algorithm is the underlying core that passively governs this decentralized system. Essentially, the consensus algorithm makes sure that every new block that is added to the Blockchain is the one and the only version of the truth that is agreed upon by all the nodes in the blockchain and a unified trust is established between the nodes of the system. Presently, consensus algorithms can be classified into two major categories:

2.2.1 Proof of Work

Proof of Work (PoW) [2] is one of the firstly adapted consensus algorithms in the early stage of blockchain development. It was first introduced with the advent of Bitcoin. Since then it is one of the most prominent algorithms for public blockchains. PoW is based on an underlying fundamental i.e. to make a node accept and add a block to the blockchain, it has to solve a tough (computationally expensive) puzzle which is easily verifiable when provided the proof. Formally we can define the problem statement and steps as follows:

Let, N be the set of the nodes participating in the network. Proof of work makes

sure that a honest node will only broadcast the block which will be validated and verified by rest of the network. The proposing node broadcasts transaction tx to the network and following steps takes place:

1. $tx \rightarrow N$ (`Broadcast`)
2. Each node collects the transactions into set of non-conflicting transactions forming a block (`Forming Block`)
3. Work on solving system set problem to provide proof pf (`Proof-of-Work`)
4. If a node $n \in N$ formulates proof pf
   (a) $n(pf) \rightarrow N - \{n\}$ (`Send` pf `to peers`)

Nodes i.e. $N - \{n\}$ accept the block only if all the transactions in the block are valid and not already spent. On acceptance, the block is created and appended with the previous hash of the chain and the system continues. This system works efficiently with open environments and every node has its own local state of validation, i.e. no node interacts with each other for proof validation. Hence if a majority i.e. $\geq 51\%$ has the same state, it'll be carried ahead with subsequent transactions. Limitations of PoW is that its throughput is very less. The throughput of the Bitcoin blockchain is 7 tps. It's computationally expensive as every node is working on solving a complex problem that uses a lot of time and resources. PoW is aligned towards open systems, not a good fit for consortium blockchains. Also, it's prone to a 51% attack i.e. if the majority of nodes are malign, then the system will be steered towards a fraudulent and inconsistent state.

Since PoW's development, there has been research observed in this area such as Proof of Stake (PoS) [32], Proof of Activity (PoA) [3], Proof of Elapsed Time (PoET) [8] and so on.

2.2.2 Practical Byzantine Fault Tolerance

With the strongly increasing needs of blockchain technology in industries, lead to consortium blockchains and with the newest adaption in consensus algorithms came

into picture i.e. using Practical Byzantine Fault Tolerance (PBFT). To understand PBFT one must know what faults it's trying to tolerate. For this, there is a conundrum called Byzantine Generals Problem.

2.2.2.1 Byzantine General's Problem

In Figure 2.2.1 we can see that there's a kingdom surrounded by an Army from all 4 sides. To capture the kingdom, the army needs to do a successful attack i.e. the army attacks on the kingdom all at the same time, else they lose. Every army has its general that gives orders to their men.

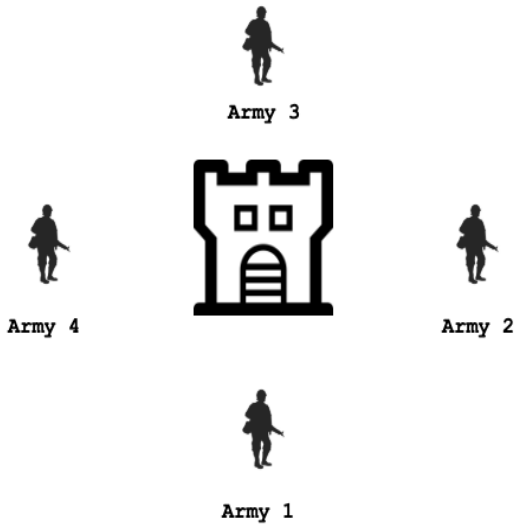

Fig. 2.2.1: Byzantine General's Problem Setup

To make sure the attack is successful one has to make sure that all General's are coordinated at the proper time. To make sure all army troops are coordinated, one general might send an army man to every troop with time noted with him, which he can extend to other troops. In this case, if the messenger is caught by the kingdom watchmen and killed, the message is not sent. There's a chance that the messenger is corrupt and updates other troops with wrong information and they lose. Troops cannot do flare shots and update the other troops as it'll be noted by kingdom watchmen and they'll be alerted about the attack. So the fundamental question lies is that, **"Given the state of the system, How can individual parties find a way in guaranteeing full consensus?"**

Transferring this problem to actual computer science realm, we map characters from this conundrum as, army as node inside the system, attack on kingdom as a task or problem statement, messengers as system messages which are distributed within the system and failure situations as arbitrary failures such as bottle-necked system delays, hardware breakdowns, change in internal state, etc. Therefore, given a distributed system, which is decentralized and trying to achieve a common, consistent and valid state, we can prevent the system to be prone to arbitrary failures using PBFT.

2.2.2.2 Working of PBFT

PBFT [7] works in 4 stages, i.e. **Pre-Prepare, Prepare, Commit and Reply**. Let N be the set of nodes i.e. where there's always one Primary Node (P) and rest all acts as replicas ($r_1, r_2, \cdots r_{n-1}$). For every PBFT iteration, P is changed which makes process more democratic.

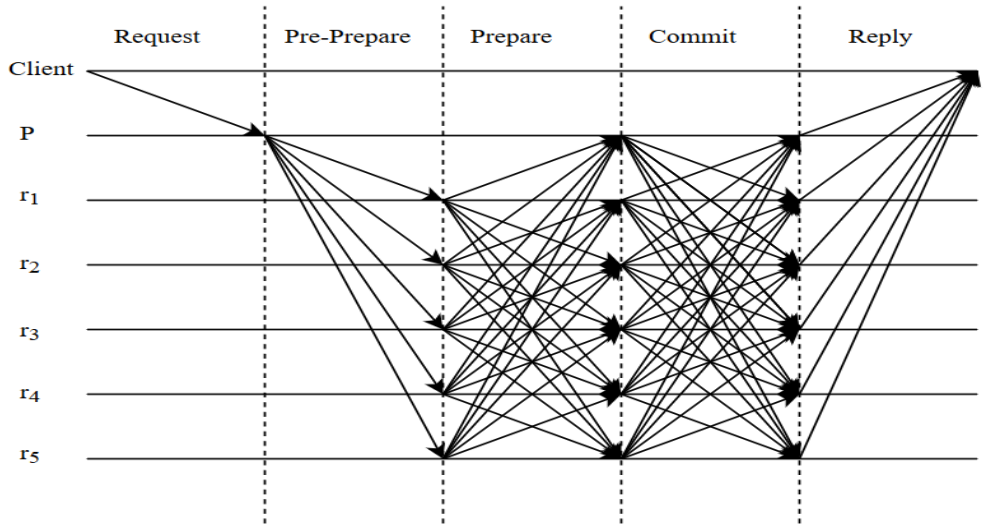

Fig. 2.2.2: PBFT Communication Flow

Whenever a client submits a transaction, P accepts the request and forwards it to all the replicas which is **Pre-Prepare** stage. Replicas make sure that everyone has the same stage and every replica broadcasts this to all other nodes which ensure that the majority of all of the replicas have a consistent state. This is **Prepare** stage. After this, replicas and P processes the transactions and shared the results with each other by broadcasting the results which is the **Commit Stage**. On validating the

reply received from other nodes, all the nodes of the system reply to Client node and if Client receives the majority of executions with the same result, we say that consensus has been reached which sums up the **Reply** stage and ends one iteration of PBFT. For the given N nodes in the system, let f be the number of faulty nodes in the system. In order to make sure that PBFT tolerates f faults in the system, $N = 3f + 1$ number of nodes should participate in the consensus.

PBFT is one of the biggest breakthroughs for distributed systems and especially for consortium blockchains. But when comparing this solution and incorporating consortium blockchains, it works well when the size of N is small. As the size of N increases, the communication complexity increases exponentially i.e. $O(N^2)$ therefore, using PBFT over big sized quorums is expensive and slow. In ***Related Work*** section we discuss the scalability part of BFTs in much greater detail.

2.3 Random Graph Theory

2.3.1 Understanding Graph Theory Basics

A graph (G) in graph theory and computer science is and ordered pair $G(V, E)$ where V is the set of vertices or nodes of the graph and E is a set of edges connecting the nodes belonging to V. Let u and v be nodes belonging to a graph G. If G is a directed graph then then edge between u and v will be an ordered pair (u, v) and in case of an undirected graph it will be an unordered pair $\{u, v\}$. Example of a graph is shown in the Figure 2.3.1.

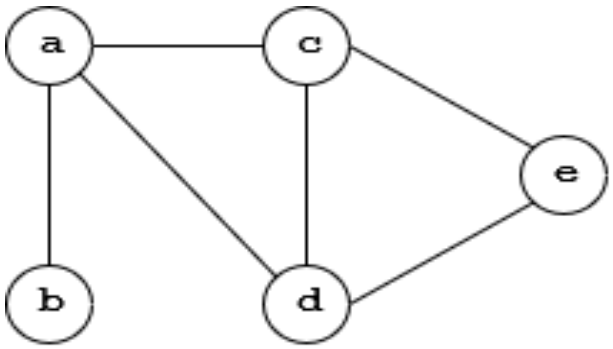

Fig. 2.3.1: A Sample Graph

For the graph shown in Figure 2.3.1,

- $V = \{a, b, c, d, e\}$
- $E = \{\{a, c\}, \{a, b\}, \{a, d\}, \{c, d\}, \{d, e\}, \{c, e\}\}$

A graph has several properties associated with it which can provide some interesting features which helps in understanding the node relationships and their roles. A node can be classified based on the way it is connected with the other nodes in the network. The parameter that defines the connectivity of the node is called degree of a node. A degree of a node in a graph is essentially number of edges incident on that node. Degree of a node is a number parameter presented as $D(x)$ where x is the node. In the sample graph, $D(a) = 3$. The neighbour set of a node is set of nodes that are directly connected to the given node. For example, neighbour set of node a is $N(a) = \{b, d, c\}$. One of the major property when relating graphs to network node connections is the articulation point. An articulation point is a vertex in the graph, when removed, leads to increase in number of connected components or leads to disconnected graph, then we say that that vertex is called articulation point. In our sample graph (Fig. 2.3.1) vertex a is an articulation point because on removal of a and its associated edges leads to two graphs (two components).

One can say that a Graph is the mathematical modeling of networks of entities connected with a certain relation. The relation between entities can be presented in the form of an edge and entities can be presented as nodes. In Computer Science, a graph is a data structure that has successfully mapped many real case problems into graph theory problems. Using results and algorithms associated with the graphs many real-world problems have been tackled and resolved.

2.3.2 Random Graph

Random Graph [6] theory is an extension to the graph theory with the addition of probability theory. Essentially, a random graph can be stated as a resultant network of events over a given probability distribution. Because of this, random graphs have

been heavily used in modeling complex networks where multiple agents are interacting with each other under a stochastic process. A complex network is a graph, which has non-trivial topological characteristics, i.e. those features are not observed in the lattice and simple graphs but are vividly observed when modeling real-time systems, such as social-networks, computer-networks (Internet), biological networks, etc. and in this case **blockchain**.

A random graph always starts with isolated nodes and edges are formed in between the nodes in a random way which is resultant of some process or computations which cannot be determined with a full assurance. Every edge in the random graph has some probability associated with it. Satisfying the probability constraint in the process leads to an edge creation between the nodes.

When considering modeling a complex network mathematically to generate a random graph and derive it's hidden properties, it looks simple because it is just nodes and edges, but the prime idea that can help one model a complex network if one knows where to keep the edges i.e. node relationships and how many of them we want to have a sufficient relationships to model the complex network. This complexities are addressed by two very famous Random graph models, i.e. $G(N, L)$ & $G(N, p)$ model, both defined by Erdos and Renyi [15].

1. $G(N, p)$ model

   This model states that each pair of N nodes in the network are connected with each other with probability p.

2. $G(N, L)$ model

   In this model, N nodes in the network are connected by L number of randomly selected edges.

Every iteration of a $G(N, p)$ graph will give a different resultant graph with changed adjacency and number of links/edges L. So it is quintessential to get an estimate how many links can we expect from a given $G(N, p)$ graph. The probability that a Random network has L links is product of following terms:

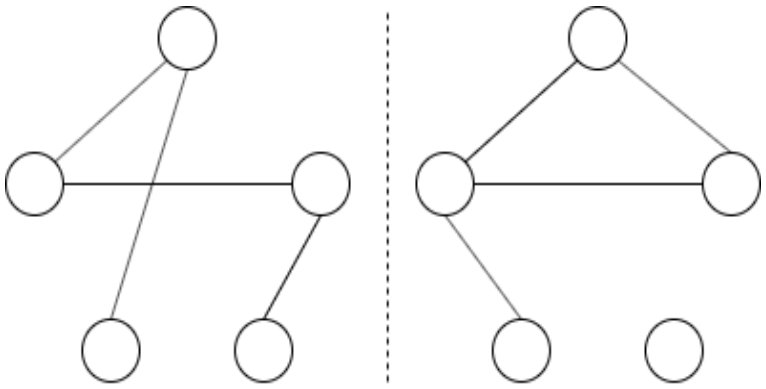

Fig. 2.3.2: $G(N, L)$: $G(5, 4)$ graph

$$p_L = (p^L)(1-p)^{\frac{N(N-1)}{2}} \binom{\frac{N(N-1)}{2}}{L} \tag{1}$$

where the terms are explained in the respective order below

1. Probability that L number of the attempts to connect the $\frac{N(N-1)}{2}$ pairs have resulted into an edge i.e. p^L

2. Probability that remaining links other than L have not resulted into an edge in the random network is $(1-p)^{\frac{N(N-1)}{2}}$

3. The combinational term, $\binom{\frac{N(N-1)}{2}}{L}$

Networks are very simple to model because it is just a matter of fact of taking nodes and edges and connecting them. The crucial part is how the links are placed which can entertain the working of an actual system. We take $G(N, p)$ model at the base for the blockchain topology design because, firstly, the probabilistic paradigm can help model the byzantine faults happening in blockchain consensus process and on the other hand with the mathematical properties of random graph we assert that communication in the consortium blockchains can be optimized to a point where a majority consensus can be reached with high probability.

2.4 State Machine Replication

In the present state of computing systems, there is a high need that the computing infrastructure is scalable and resilient. The scalability of a computing system is defined based on the performance and execution time taken by the system to execute the compute query. On top of that, resiliency is an important factor to keep system fault-tolerant and prone to failures. Existing large-scale systems needs to accommodate these requirements.

State Machine Replication (SMR) [33] paradigm in client-server architecture involves creating a robust, fault-tolerant service by replicating servers and orchestrating client requests with those server replicas. In distributed systems, where multiple servers are working with each other serving different client requests at once needs to make sure that they have the same system state i.e. data to serve. SMR plays an important role to make sure that all the nodes in the system maintain a consistent and valid state.

A SMR system consists of following elements:

- Set of Nodes N

  A set of nodes is essentially a set of entities participating in the system where over a series of processes, the state of the entity might get changed. It can be a file, a compute node, database, etc.

- Set of States S

  A state is a collection of internal parameters that determines the condition of a node. A global set of all such possibilities of given parameters for a given process is called a set of the set of states.

- Set of Inputs I

  A global set of instructions specific for a process which can be passed to a node for processing is called the set of Inputs.

- A Transition Function $T(s, i)$

  A transition function is essentially the underlying logic, running on a node to validate the input on a given state of the node. A transition function is passed with two parameters i.e. s and i where where $s \in S$ and $i \in I$. Based on the parameters passed to a node, a node may change a state its state or retain it.

Every node in the system has a state s associated with itself related to an operation. A state s is a set of parameters defining properties of a node that are relevant for the operations that it performs. A node may receive an input i which is evaluated over the node's state s by a transition function T leading to a new state $s^{'}$ or can simply just lead to an output which involves no state update. This can be formally represented as:

Let N be the set of nodes and I be set of inputs. A node $n \in N$ receives input $i \in I$ and state update function can be defined as follows:

$$T\Big(n(s), i\Big) \Rightarrow \Big(n(s^{'}) \ \vee \ n(s)\Big) \wedge Output \tag{2}$$

where $n(s)$ represents a state s associated with a node n

CHAPTER 3

Related Works

3.1 Approaches to Scale Blockchain System

With the realization of the potential that blockchain technology can bring into the industry, there has been avid research observed in this area to make the technology efficient, secure and effective with regards to replacement for the existing technology stack. Out of all the areas, **scalability** of blockchain systems is a pivotal task because the scalability of the system is directly proportional to efficiency and adaptability. [39][25] Scaling Byzantine consensus is a modular approach where different aspects of the consensus mechanism are taken into consideration and optimized based on the need. Our recent survey shows that while architecting a blockchain, Quorum Election, Communication Topology, Cryptography techniques and Trusted Execution Environments (Specialized Hardware Technologies) are prime factors leading to scaling of Byzantine consensus when applied in various phases of consensus. The current research uses an amalgamation of these techniques to formulate new solutions that are efficient and industry ready for adaptation.

3.1.1 Quorum Formation

Scalability gets hindered in state of the art Byzantine consensus algorithms where the entire network participates in processing and validating a single transaction. Making the whole network to verify the transactions not only increases the complexity in achieving the consensus but also leads to a high amount of communication overhead. So the notion behind the quorum formation is to use a subset of nodes to validate

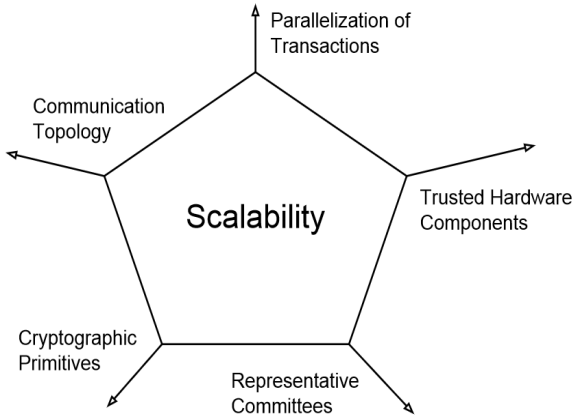

Fig. 3.1.1: Approach to Scalable BFT Design [4]

the transaction and the rest of the nodes in the network learn from them. This can be formally represented as follows.

Let N be the complete set of nodes in the blockchain network. $N^{'}$ number of nodes out of N forms the consensus committee which is responsible for running consensus among them and remaining nodes, the learner nodes learn from the consensus committee. Here the learner nodes have no interaction with the consensus committee which makes learner nodes a part of passive replication schemes as they accept the results computed by $N^{'}$. This essentially reduces the resource foot-print and communication complexity of the system as only a subset of nodes have to communicate with each other.

This approach was first used in CheapBFT [20] and ReBFT [12] where a secure Field Programmable Array (FPGA) based subsystem of nodes was responsible for validating transactions and rest of the nodes were part of passive replication. Instead of $3f+1$, these approaches used $2f+1$ active replicas and the remaining f are passive, but there are assumptions. The system will only rely on $2f+1$ replicas during normal state (no byzantine faults or arbitrary failures). In case of failures, fall back protocols are activated which again uses $3f+1$ nodes.

3.1.2 Communication Topology

Communication topology governs the way nodes interact with each other in the system. It can be static or dynamic depending on the implementation of the system. State of the art PBFT does not scale well due to heavy communication overhead it takes (Figure 2.2.2). In PBFT at every phase of the process, every node in the network exchanges message which makes it more complex.

The most optimal and least complex communication topology observed in the literature is FastBFT [26]. FastBFT uses a balanced tree formation within the network, where the leader is at the root of the tree and then based on the desired branching factor (b), other nodes are aligned. Messages are propagated from top to bottom and leaf nodes reply and intermediate root nodes collect and aggregate messages to the top. The use of a tree as communication topology reduces the communication complexity but it compromises with the liveliness of the system. In the case of arbitrary node failure in a tree, the entire subsystem connected with that node will be unavailable. Bigger the height of the node more adverse the failure becomes. The communication complexity of this system is $O(b)$.

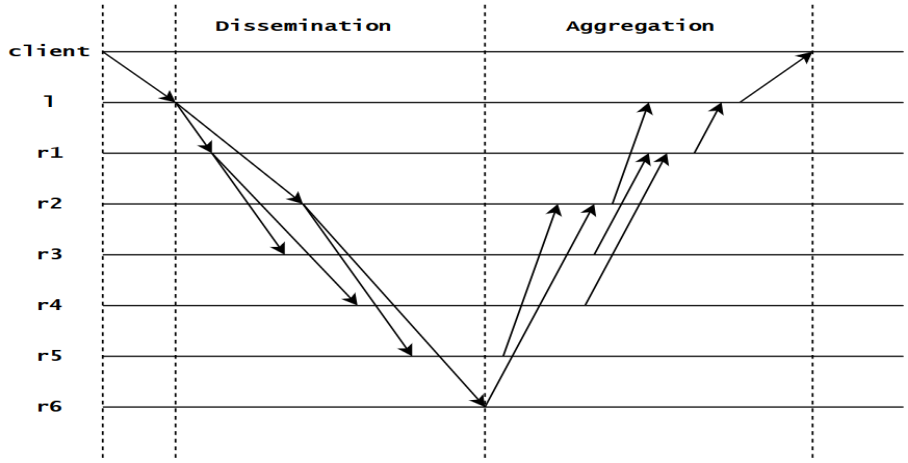

Fig. 3.1.2: FastBFT Communication Flow with $b = 2$

LinBFT [38] presents a paradigm to cut short $O(n^2)$ communication complexity of PBFT to $O(n)$ by aggregating all the results from the network in every phase. Instead of nodes interacting with each other, the node sends cryptographically signed

messages to the leader and the leader validates the messages. This approach bottlenecks as a leader has to aggregate, verify and disseminate the messages to the rest of the network.

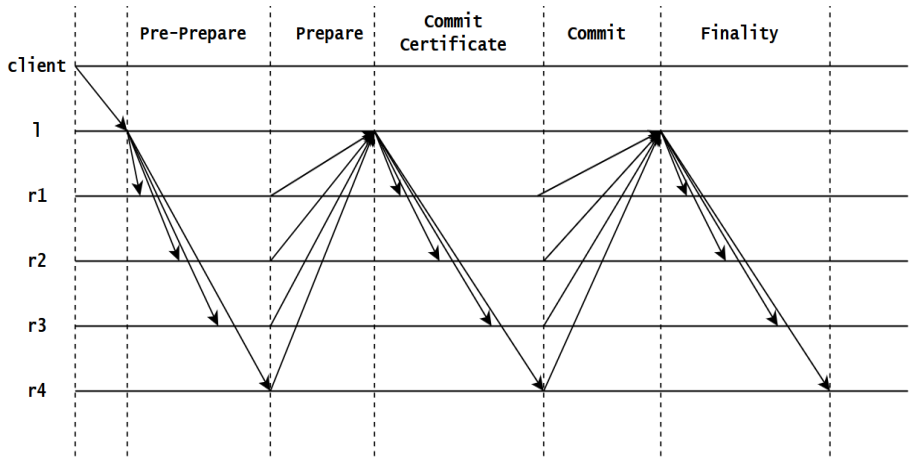

Fig. 3.1.3: LinBFT Communication Flow

A similar but more optimistic rather speculative approach is used in Zyzzyva [22] where once the leader submits the transaction to the nodes in the network, all the nodes speculatively validate and execute the transaction and return the result back to the client instead of the leader. Client on finding inconsistencies in execution among the nodes corrects nodes converging to a common state of the system, reducing replication cost to theoretically null (in absence of byzantine nodes). But instead of having multiple phases, the gracious execution of Zyzzyva only takes 2 phases.

3.1.3 Cryptographic Techniques

Cryptographic primitives have played an essential role in reducing computational and communication overheads in achieving the consensus. Cryptography and reduced computation might sound oxymoronic, but it has substantially helped in many different ways. Algorand [16] in its cryptographic sortition makes use of Verifiable Random Functions [28] (VRFs), where every node in the network runs the function to know his role in the consensus. Not only does this eliminate the communication overhead between the nodes with regards to the selection of a committee to initiate

consensus but also makes leader and committee election free from any malign activities adaptive adversaries can possibly do to disrupt the process. Gosig [24] uses a similar cryptographic leader election scheme like algorand with a multi-round voting mechanism. The method also tries to optimize the communication by frequent dissemination using a gossip-based methodology but is not optimal with regards to the regards to dissemination scheme.

Advanced signature schemes have also played an essential role in reducing authentication payload by trimming signatures of multiple nodes form of a committee to a single signature. One of the prominent use-cases is in FastBFT where Collective Signing [34] (CoSi) method is used to construct Schnorr multi-signature [27] . CoSi is used in FastBFT's message aggregation phase where the quorum forms a balanced tree having leader node at the root and messages start aggregated from leaf nodes and at the end root has a single message signed by all the nodes of the quorum.

3.1.4 Trusted Hardware Components and Trusted Execution Environments

Recent research shows the use of trusted hardware components and secure computing schemes to reduce the computational resources used in validating the operations within the peers. One of the earlier use-cases was observed in the ReBFT and CheapBFT. In these protocols, a smaller subsystem of nodes essentially decided the state of the system as the rest of the nodes simply replicated the results formulated by the consensus committee, to make sure the subsystem is not malicious, FPGA based trusted system is incorporated to authenticate and verify the protocol messages.

A much more enhanced approach is observed in FastBFT which uses Intel SGX [9] service provisioned by CPU for creating a secure Trusted Execution Environment [31] (TEE) which is immune to sybil attacks and tamper-resistant from any other system calls happening inside the node. FastBFT leverages an optimized lightweight secret sharing scheme over TEE with efficient message aggregation protocol making it one of the most scaled byzantine consensus protocols in the literature. However, using

specialized hardware schemes in the Byzantine consensus is also a constraint in terms of acceptance and incorporation in the existing system as a ready to go functionality.

3.1.5 Off-Chain Paradigm

Off-chaining has been a new area of research observed in blockchains which has the potential to enhance scalability and privacy of the system. Off-chaining focuses on outsourcing the data storage (not the ledger) and computations to a third party trusted subsystem without sacrificing the fundamental essence of blockchains i.e. **immutability** and **availability** of data. This research is especially relevant to public blockchains as every transaction processed in the system is evaluated and executed on every node in the network, leading to less throughput. And as far as user-specific data is concerned, keep data available on all the nodes of the system does not guarantee the privacy of the data. Off-chaining has capabilities to extirpate the limitations expressed above.

3.1.5.1 Off-Chain Storage

It has been pointed out that with increasing use-cases of the blockchain technology, storing data on-chain becomes expensive. The data includes essentially assets and tokens representing the data directly related to users. To validate the transactions, not only the ledger but also this user-specific data needs to replicate over the entire network. In public blockchains, this leads towards reduced privacy over the personal data and the chance of data leaks is high.

The way to address this issue is to export the user-specific data to a highly available secure off-chain storage system capable of handling failures. Whenever blockchain has to execute a transaction that involves state change in the data, first the blockchain network retrieves data from the highly available storage ensuring the integrity of the data. After retrieval, the transaction is processed and state update on data is performed and finally, third party storage writes changes on the data and updates the state. Here system relies on the third party to update the data on the

storage system. To make sure that changes are made properly and there is no malice in the third party system, a content-addressable storage system [14] can be taken into consideration. This can by following steps shown below while accessing the data:

1. Each data-file (d) is stored in the content-addressable storage and mapped with its hash-value ($hx(d)$).

2. User stores reference address of d i.e. $ref(d) = hx(d)$, and keeps $ref(d)$ privately mapped in it's smart-contract

3. While querying the data, user can use $ref(d)$ to access d. On retrieving d it can calculate the hash again and verify the integrity of data.

This approach solves secure out-sourcing data to a third party storage system but the problem prevailing with the liveliness of the system still prevails. In case of data loss, blockchain cannot do anything to retrieve the data back. Hence, for now, the off-chaining approach for exporting data seems an active research area needing substantial and strong improvements encompassing the liveliness of the blockchain system.

3.1.5.2 Off-Chain Computing

Off-chain computing is a paradigm in which the execution part of blockchain gets outsourced to an off-chain network and verification part is performed on-chain [13]. Suppose time taken to perform an on-chain computation be t_{on} and off-chain be t_{off}. And let time taken to verify the computation be t_{ver}. Off-chain computation guarantees that $t_{off} <<< t_{on}$. But the complete notion is useful when, $t_{off}+t_{ver} < t_{on}$

Off-chaining introduces trust issues because consensus computation is dependent on the untrusted third-party. The crucial part is the verification phase because the resultant computation of the off-chain network determines the next state of the blockchain. So if the off-chain network successfully supplies false verification proofs, the chain state gets corrupted. This approach reduces the computational overhead but maximizes trust issues.

To solve this quest, verifiable computation schemes can be used which use cryptography in its base which can improve the trust in the verification phase with high magnitudes. The verification algorithm used in this approach should be cheap making t_{ver} less and algorithm should be non-interactive, i.e. the prover should prove its computation in a single message or with very few message exchange leading to reduced communication overhead. Zcash [19] [40] uses verifiable computations to validate the transaction in their blockchain. They use Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (ZkSNARKs) [36], which is a special kind of zero-knowledge proof [17] which lets a prover prove its computation within a fraction of seconds, and the proof size is also minimal. However, the initial setup phase, which is a one-time setup is computationally expensive than the actual execution of the computation.

3.2 Gossiping in Distributed Systems

Gossip based algorithms also known as epidemic algorithms were initially developed to reliably propagate a set of data over a group of nodes forming a distributed system, where the data shared essentially defines the operations or state update messages. The simplicity, robustness, and effectiveness have made gossiping algorithms an integral part of large scale distributed system design [18]. Systems that use gossip have the same underlying design with some tweaks and differences in algorithmic details based on the use-cases that divert the working of these systems.

Gossip in a distributed system means disseminating information inside the network by selecting nodes at random and passing the information to them which will eventually populate the entire network with the same information. The initial stage of information dissemination is slow, but after a couple of iterations, the spread of information is at an exponential rate. The first instance of usage of gossip in distributed systems was used to keep the database consistent in replicated database designs [11]. Since then, technology has come a long way and in recent developments, gossip/epidemic algorithms have been widely used in Wireless Sensor Networks, Ad-Hoc Systems, large scale distributed systems which has a strong requirement of fault-tolerant

computations and rapid information exchange. A general organization of gossiping protocol is as shown in the figure below.

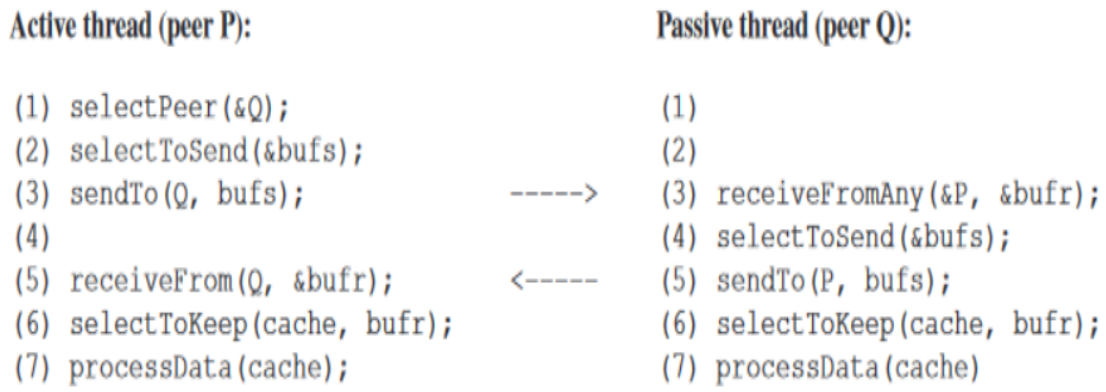

Fig. 3.2.1: General Structure of Gossiping Algorithm [21]

A node/peer P selects a node at random (Q) from available list of nodes in the network. Based on the state of current data P selects what data needs to be shared (bufs). P sends bufs is sent to Q. All the nodes in the network are continuously accepting messages and in mean time Q receives message from P. In meantime if Q has any new data to share it shares it's data with P. Regardless of previous step, Q checks the state of received data with its cache and retains the new data and updates the cache, which also happens at the other end in P. A node can contact one more more nodes at a time, which determines the span of a node in the system. This parameter is called **fan-out** parameter (m). A node can also be controlled how many times it can disseminate the messages for a single process. This two parameters define the connectivity and relationship between different types of nodes in the network.

CHAPTER 4

Methodology

To tackle the problem statement, we keep the consensus execution restricted to the validating blockchain quorum and extend the logic to interpret the finality if consensus to the rest of the quorums who are associated with the shared data which is being validated by the quorum who is running the consensus which will serve as a base for a consistent state of data over the network with **state machine replication**. Dissemination of the message from validating blockchain to rest of the blockchain in a decentralized fashion is done in an optimized communication topology that is resultant of **informed gossip** over the **random graph**. To elaborately explain the approach we divide the entire process into two phases.

1. Consensus Phase

   In this first part of the process, the blockchain client assigns transaction block to one of the organization as per the request transactions have made. Blockchain client decides who will be the leader to initiate the consensus. The organization that got selected runs the consensus mechanism C and this phase comes to end.

2. Dissemination Phase

   Here once consensus in the phase one is reached, dissemination phase happens on the **residual network**. Residual network is network consisting the leader node of consensus achieved organization and all the nodes belonging to the rest organizations sharing the same data. The residual network can be formally represented as follows:

Let there be n number of organizations over a global set $O = \{O_1, O_2, O_3, ...O_n\}$ and O_i was the one to initiate the transaction in consensus phase. Let l be the leader node from O_i. Therefore residual network $N^{'} = (O - O_i) \cup l$

4.1 Node State Parameters

Every node in the system is part of state machine replication and has a state associated with itself. The state of node is defined by following parameters:

- *db*: The data which is being handled by the node
- *tx_index*: Indexed list of the transaction already processed and updated in the ledger of the node
- *dis_counter*: This defines the number of times a node has disseminated a particular transaction over the random graph

This parameters are updated by the transition function called $updates(tx, f^*)$ where tx is the transaction which needs to be processed on node and f^* is the proof of consensus validation from validating blockchain w.r.t tx. This is outlined in the Algorithm 4.1.1

Algorithm 4.1.1 updates(tx, f^*)

Input: tx which is transaction ; f^* is finality of consensus associated with tx
Output: State update of the node

```
if tx validates f* then
    update db
    update tx_index
    create block for tx
else
    reject tx
end if
```

4.2 Informed Gossip

Informed gossip [23] is the integral part responsible for reducing the overall communication overhead of the system and responsible for generating the random graph with the nodes from the residual network. In this process every node holds an index of a set of nodes who has process a particular message and when they gossip, with the message they also share the set of nodes from the index for that particular message. When a node receives the message, it compares it with it's local set and updates it with the new entries. A logical view of a node index is as shown in Table 4.2.1.

Message	**Informed Node Set**
m_1	$\{d, a, c\}$
m_2	$\{b, e, c\}$

Table 4.2.1: Logical View of Node Index

To illustrate this process let's take an example of a 5 node system i.e. $\{a, b, c, d, e\}$ shown below for a single message m_1 where a is the originator of the gossip. For this example we set the fan-out parameter of gossip to be 2 i.e. every node can send message to 2 nodes in an iteration.

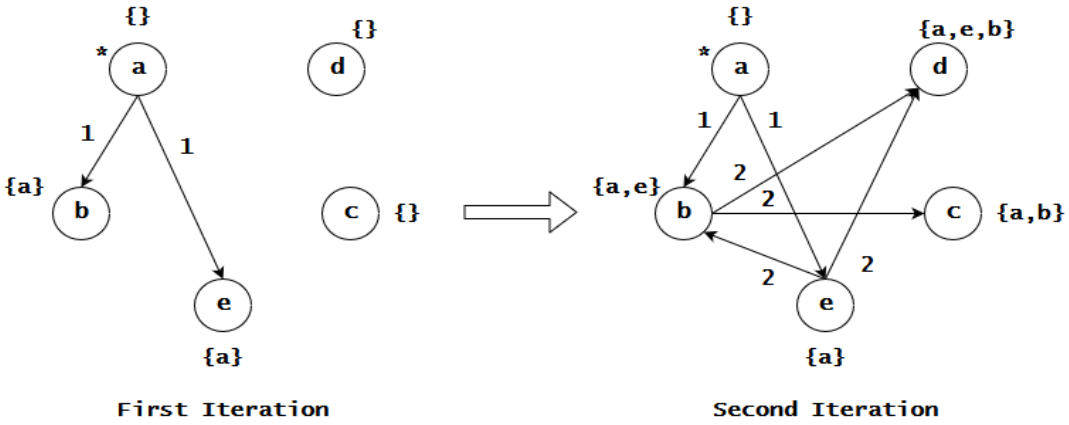

Fig. 4.2.1: Informed Gossip Example

As shown in the Figure 4.2.1, all nodes start with an empty node index. Node

a receives the message and based on the defined fan-out parameter, it chose b and e at random. Now b and e know that a already has m_1, so for m_1 the node index of b and e becomes $\{a\}$ and first iteration comes to end. In second iteration, b and e gossips m_1 with their node indexes. At the end of second iteration we can see that d has it's state updated to $\{a, e, b\}$ based on the node indexes received from e and b and node c has its node index updated to $\{a, b\}$ based on node index of b. And this is how informed gossip works. One can note that one can optimize the process by adjusting the fan-out parameters and terminating conditions of when gossip should end. The fan-out parameter will define the reach of a node and terminating condition will determine how many times at most a node will disseminate the message in the network.

In the formal outlined solution we achieve the informed gossip with the function disseminate(tx,f^*, $cache[tx]$) where the $cache[tx]$ is the node index of tx. We restrict to a node to trigger disseminate(tx,f^*, $cache[tx]$) for itr times and for the gossip we have fan-out parameter set to $ln(N)$ where N is the global set of nodes from residual network. The outlined function is shown in the Algorithm 4.2.1.

Algorithm 4.2.1 disseminate(tx, f^*, $cache[tx]$)

Input: transaction tx, f^* finality of consensus for tx and local cache of node that consists of all the other nodes that processed tx i.e. $cache[tx]$
Output: disseminates the transaction to other nodes in the system

```
if dis_counter[tx] < itr then
    N' = select ln(N) nodes at random from N
    (tx , f*, cache[tx]) → N'
    dis_counter[tx] += 1
else
    continue
end if
```

The reason behind using $ln(N)$ as the fan-out parameter can be derived from a fairly simple random graph results and proofs. If we take a look at equation (1) in chapter 2 in subsection 2.3.2 of section 2.3, the probability that a given $G(N, p)$ graph has L links is provided. Equation (1) is essentially a binomial distribution and hence the first moment of this equation will give us the mean value for L i.e. $\overline{L}$ which is an

estimate of number of links in the graph i.e.

$$\overline{L} = \sum_{L=0}^{\frac{N(N-1)}{2}} L \cdot p_L = p \cdot \frac{N(N-1)}{2} \tag{1}$$

For a given graph G, average degree of a graph (k) can be given by following equation:

$$k = \frac{2E(G)}{v(G)} \tag{2}$$

Extending this equation to random graph will give us:

$$k = \frac{2\overline{L}}{N} = p(N-1) \tag{3}$$

In the normal case operation for N nodes in a $G(N, p)$ model, for a high value of p as the system progresses all the isolated nodes converges into a giant component. With a justifiable high value of p we want to make sure that the giant component at the end of the process has all N nodes in it. For that we need an estimate that what should be the value of k because for given value of p, k defines the span and connectivity of the nodes in the graph. So the condition that we want is that number of nodes in the giant component (N_G) to be approximately N i.e. $N_G \simeq N$.

Therefore, probability that a randomly selected node is not linked with the giant component is

$$(1-p)^{N_G} \approx (1-p)^N \tag{4}$$

Hence, expected number of such isolated nodes (I_N) are

$$I_N = N(1-p)^N = N(1 - \frac{N.p}{N})^N \tag{5}$$

The fraction term in the equation (5) can be approximated in terms of e i.e.

$$(1 - \frac{x}{n})^n = e^{-x} \tag{6}$$

Using equation (6) in (5) we get,

$$I_N = Ne^{-Np} \tag{7}$$

Assuming there is one isolated node in the network deprived from any connectivity to N_G, i.e. $I_N = 1$, we get

$$Ne^{-Np} = 1 \tag{8}$$

$$ln(N) - Np = 0 \tag{9}$$

Finally for any value of N, we get,

$$p = \frac{ln(N)}{N} \approx 0 \tag{10}$$

Using result from equation (3) we can rewrite equation (9) as

$$k = pN = ln(N) \tag{11}$$

And hence, for $N_G \simeq N$ to be closely equal over respectable high value of p we need $k = ln(N)$. Therefore, for any random graph to be a giant component with all nodes covered as part of it's stochastic process, the average degree of a graph needs to be $ln(N)$. This is called critical point in the random graph building process, a point of convergence [29].

4.3 Proposed Solution

For the system process, we have nodes continuously listening for the messages in the network. Accept the incoming message and validate it. Nodes follow a set of instructions based on the events that take place in the process. In this case, we have two kinds of events one is when a timeout occurs for a message and second is when a node's active cache (a_cache) of message for a particular transaction is full.

The active cache is a memory allocation, where a node accepts a certain number of messages of transaction messages in the period and if in the period the cache gets full, node disseminates else for that transaction it will disseminate on the timeout. Based on this an outlined process is shown in Algorithm 4.3.1.

Algorithm 4.3.1 SMR with Information Dissemination

Input: Node is listening for a tuple (tx,f^*,$cache[tx]$) over the network which is input to the system

Output: Event triggers

```
while True do
  e ← Event()
  if e is TIMEOUT then
    if tx not in tx_index then
      updates(tx, f*)
      disseminate(tx,f*, cache[tx])
    else
      disseminate(tx,f*, cache[tx])
    end if
  else
    continue
  end if
  if e is ACTIVE_CACHE_FULL then
    cache[tx] = cache[tx] ∪ a_cache[tx]
    N = N − a_cache[tx]
    a_cache = φ
    if tx not in tx_index then
      updates(tx, f*)
      disseminate(tx,f*, cache[tx])
    else
      disseminate(tx,f*, cache[tx])
    end if
  else
    continue
  end if
end while
```

As shown in the procedure we use the disseminate and updates function which is responsible for message passing efficiently and state update of node respectively. This procedure runs continuously at the background in the node listening to the messages in the network and based on the events triggered, node functions in that procedure.

CHAPTER 5

Experimental Results & Analysis

Based on the methodology discussed we develop the system and test it various scenarios and derive the analysis of the results. This chapter is divided into complexity analysis, empirical analysis, system analysis and comparison with other methods.

5.1 Complexity Analysis

If we observe the procedure established in Algorithm 4.3.1, every node in the system will disseminate for at most itr times. The fan-out parameter we have for the nodes for every iteration stays constant i.e. $ln(N)$ for a given value of p. So essentially for each node communication overhead is of the order $O(itr \cdot ln(N))$. Therefore, for the overall N nodes in the system, the total cost over communication is in the order of $O(N \cdot itr \cdot ln(N))$.

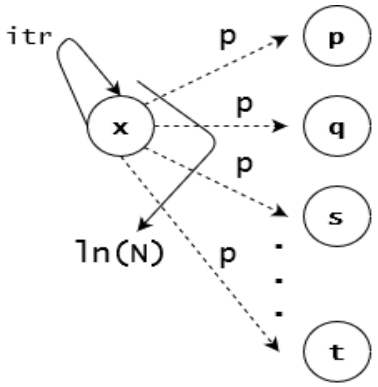

Fig. 5.1.1: Logical View of Node Dissemination

5.2 Empirical Analysis

The empirical analysis is quintessential for this model because it essentially helps to determine the properties and traits that the system would generate when it is functioning regularly. Using empirical analysis we target the random graph which is generated when nodes interact with each other. This will essentially also help in determining how changes in the parameter affect the overall working of the system. We consider the connectivity of nodes and effects of p and itr as the primary motive to achieve the analysis.

5.2.1 Scalability Aspect of Model - Number of Connections vs Number of Nodes

Here, we check how the model works with the growing size of N and what is the growth of the network when more number of nodes participate in the process. For this, we run the model over the same p and itr value but keep on increasing value of N at a fixed interval and observe the number of connections in the system. For this experiment, we default $itr = ln(N)$ and $p = 0.8$. Figure 5.2.1 represents the relative change w.r.t. N.

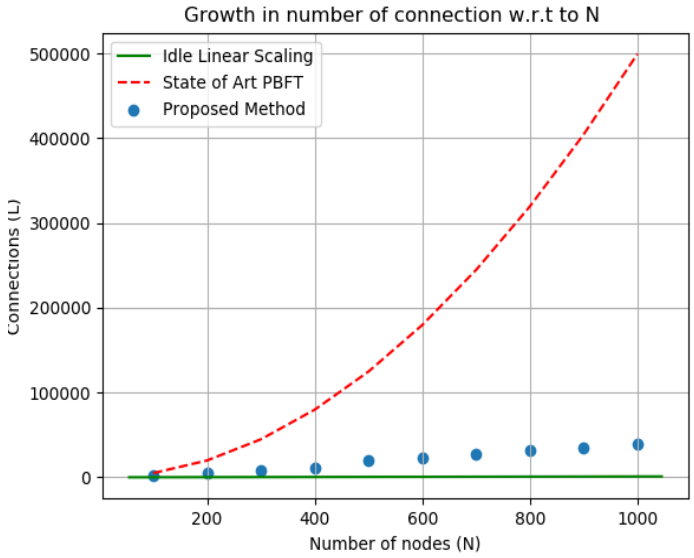

Fig. 5.2.1: Growth of Edges with increasing number of nodes

We compare the scalability with the State of Art PBFT and to have a baseline we assume an ideal mark as the linear growth function i.e $f(x) = y$ form of the line. And we see that our method works sub-optimally when compared with the ideal state and is highly scalable when compared with the communication overhead and costs associated with PBFT. This analysis states that our proposed model can scale very well when an increase in the number of nodes is observed.

5.2.2 Effects of itr and p on Model

In this analysis, we observe the changes made by itr and p and how it affects the model. p is essentially a probability measure that determines how responsive the nodes in the model are. Higher the value of p essentially should mean that there is less number of failures in the process. We check the model for $N = 100$ nodes over $itr = ln(N)$ for p value set $\{0.5, 0.6, 0.7, 0.8\}$ and results are presented in Figure 5.2.2 - Figure 5.2.4.

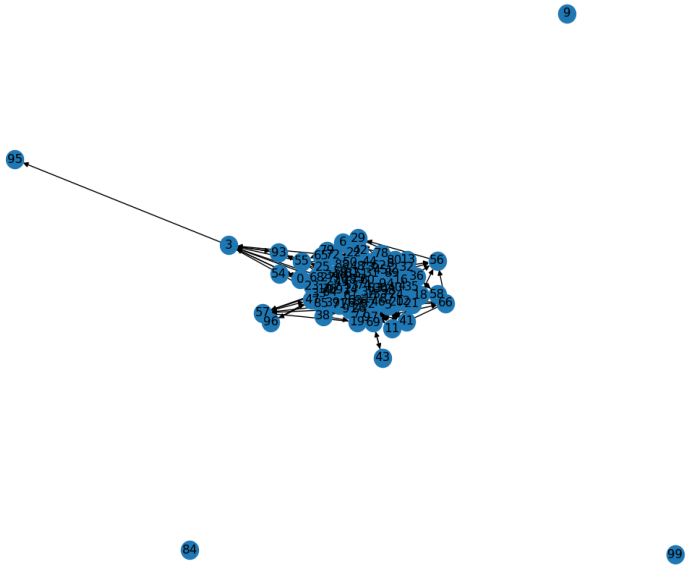

Fig. 5.2.2: $G(100, 0.5)$ with $itr = ln(N)$

As shown in figure 5.2.2, for the low p value like 0.5, we have 3 isolated nodes and one node with literally one incoming edge. Even in such detrimental effects on the system, we still attain to achieve 94% connectivity. With the increasing value

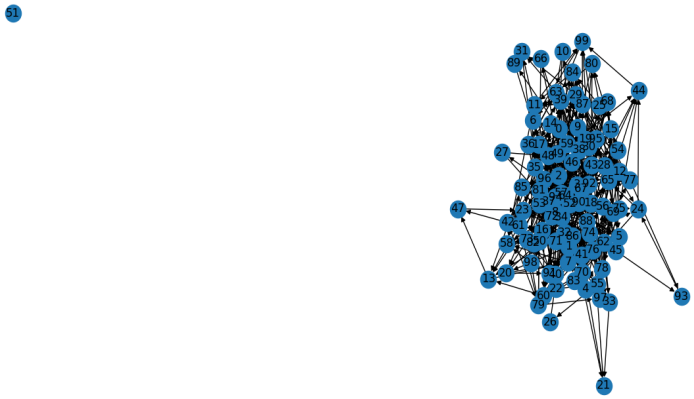

Fig. 5.2.3: $G(100, 0.6)$ with $itr = ln(N)$

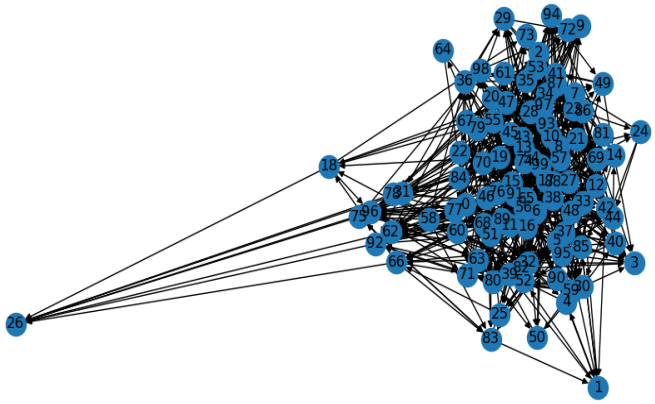

Fig. 5.2.4: $G(100, 0.7)$ with $itr = ln(N)$

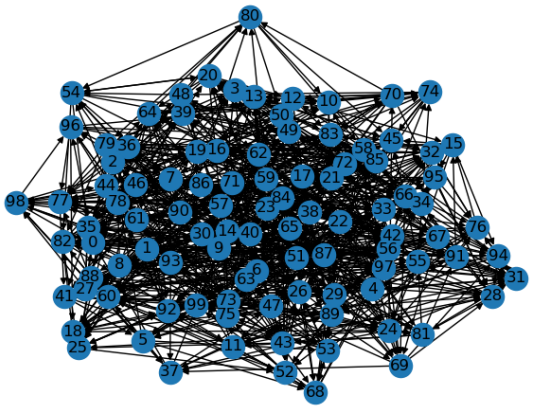

Fig. 5.2.5: $G(100, 0.8)$ with $itr = ln(N)$

of p we see less isolated nodes in the system and at $p = 0.8$ we have one giant component where $N_G = N$. So we can say that even when a decentralized system at 20% arbitrary failure rate, our model makes sure that every node receives enough number of messages to have a strongly connected network. With this analysis, we conclude the effects of p.

To understand the effects of itr we will consider a $G(100, 0.5)$ with $itr = ln(N)$. The initial result that we observe is shown in Figure 5.2.6. We can see that with default itr value for low values of p we see $N_G < N$.

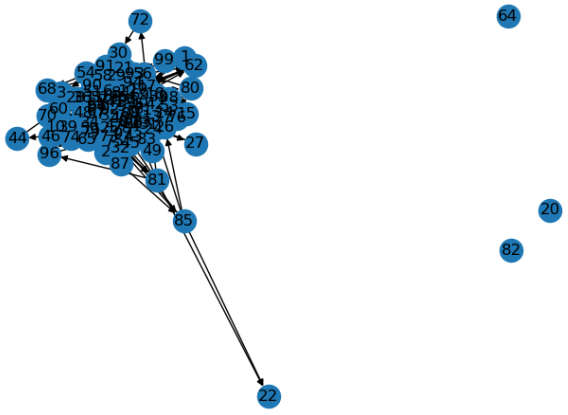

Fig. 5.2.6: $G(100, 0.5)$ with $itr = ln(N)$

For the same values p and N, with increasing itr by a unit starts making a difference in how node interacts and leads to a reduction in isolated components to no isolated components. This is shown in Figure 5.2.6 - 5.2.8. With the increasing value of itr, we observe nodes have the freedom to span to multiple nodes. With this, assuming a distributed system suffering from arbitrary failures can cope up with the increase in the reachability of nodes. Table 5.2.1. presents the numerical details of this model experiment.

Additionally, to check how the model performs over a series of experiments without changing the parameters, we run the model and test how much model varies over time. For this we run $G(N, p)$ for fixed $itr = a\_cache = ln(N)$ and simply vary p to determine the changes. For this, we iterate the model over 100 experiments and we record the attributes and mean and standard deviations for the same. A sample time

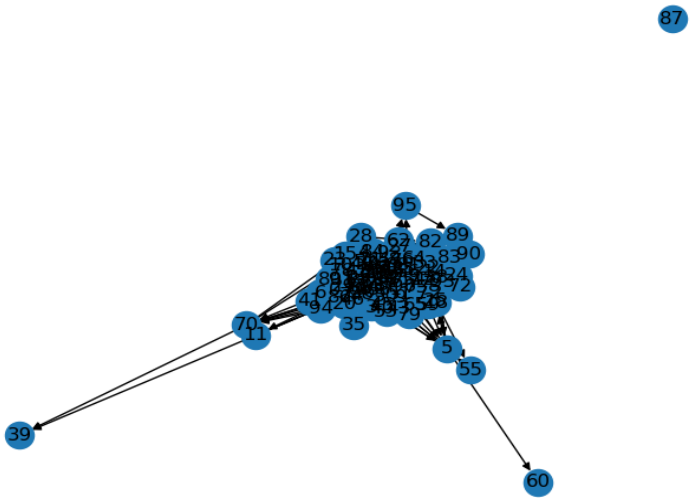

Fig. 5.2.7: $G(100, 0.5)$ with $itr = ln(N) + 1$

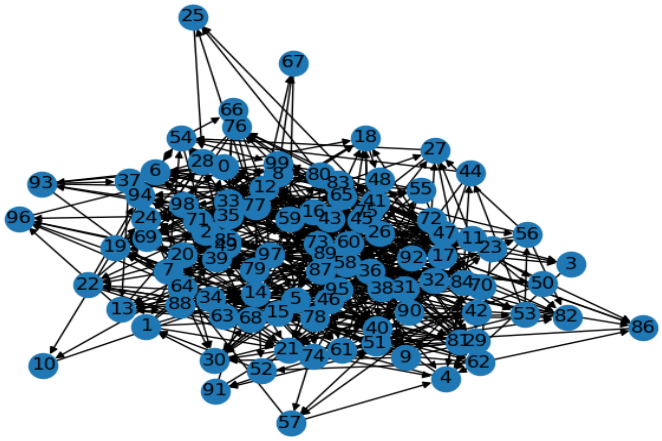

Fig. 5.2.8: $G(100, 0.5)$ with $itr = ln(N) + 2$

series plot is shown in Figure 5.2.9 which highlights a $G(100, 0.6)$ graph tested over 100 experiments with regards to changes in the number of links (L). For this test, we get the mean ($\overline{L}$) as 593.54 and standard deviation (σ) as 61.67.

Extending this experiment over other values p for 10 experiments with pre-set $itr = 2$, we evaluate $\overline{L}$, σ, min and max in of L, $\max(N - N_G)$ representing maximum number of isolated nodes in experiments, and $N_G \neq N$ situation to determine how many times isolated nodes were observed in the experiments. The details are illustrated in Table 5.2.2.

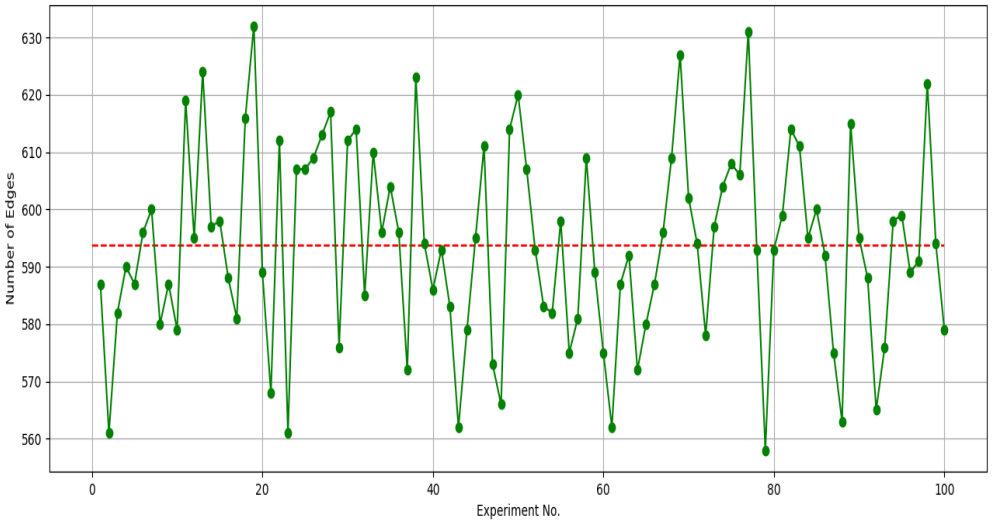

Fig. 5.2.9: $G(100, 0.6)$ over 100 experiments

$G(N,p)$	itr	L	$\lvert N - N_G \rvert$
$G(100, 0.5)$	$ln(N)$	369	3
$G(100, 0.5)$	$ln(N) + 1$	432	1
$G(100, 0.5)$	$ln(N) + 2$	539	0

Table 5.2.1: itr experiment details

p	$\overline{L}$	σ	min	max	$N_G \neq N$	$max(\lvert N - N_G \rvert)$
0.9	899.1	9.11	882	918	0	0
0.8	799.95	11.01	779	829	1	1
0.7	696.34	17.3	653	732	7	2
0.6	590.54	61.67	515	621	12	4

Table 5.2.2: Overall Results of Variation Test

Based on the values presented in the Table 5.2.2., we can say that over series of 100 transaction executions, only 12% of time, nodes were deprived of the updates, and that too only for maximum of 4 nodes, were not in N_G which can be easily

remediated by tweaking itr which we saw in itr and p analysis. Also as we expected, with increasing p value isolated nodes have been reduced. This sums up the empirical analysis of our model.

5.2.3 Degree Distribution and Connectivity of Graph

In this section, we discuss the degree distribution of the generated network and analyze its thresholds against the ideal cases. For this we have a $G(100, 0.8)$ with $itr = ln(N)$. For the generated graph we get 1038 links and detailed plots are shown in Figure 5.2.10, 5.2.11 and 5.2.12. The generated graph is presented in Figure 5.2.13.

In the given plots, a solid line is presented which represents Ideal mean of degrees for distribution to satisfy a fully connected network i.e. $ln(N)$ and the dashed line represents the average degree distribution of generated graphs. In all the cases we can see that the generated graphs exceed the baseline by almost twice as the ideal baseline for fully completely network. Also, the majority of nodes are skewed on the right-hand side (growing region) of the ideal adding a promising factor in terms of clustering of network i.e. none of the nodes during gracious executions are loosely connected.

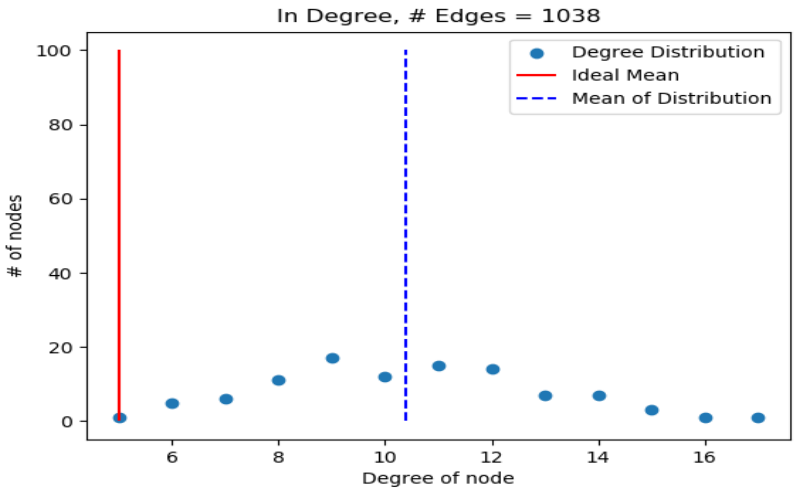

Fig. 5.2.10: Degree Distribution - In Degree

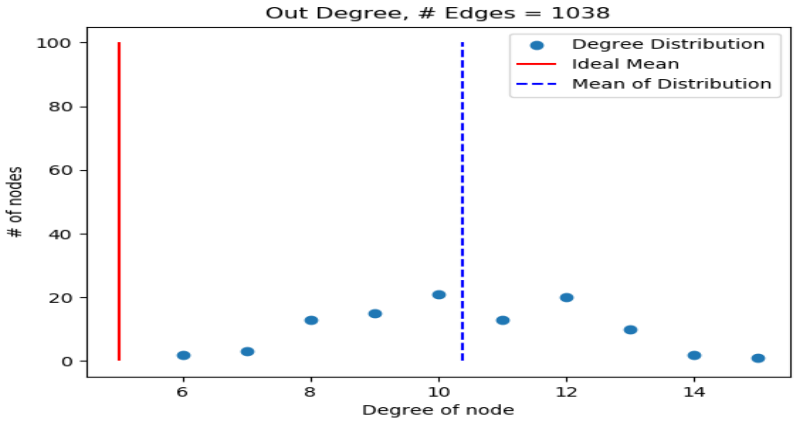

Fig. 5.2.11: Degree Distribution - Out Degree

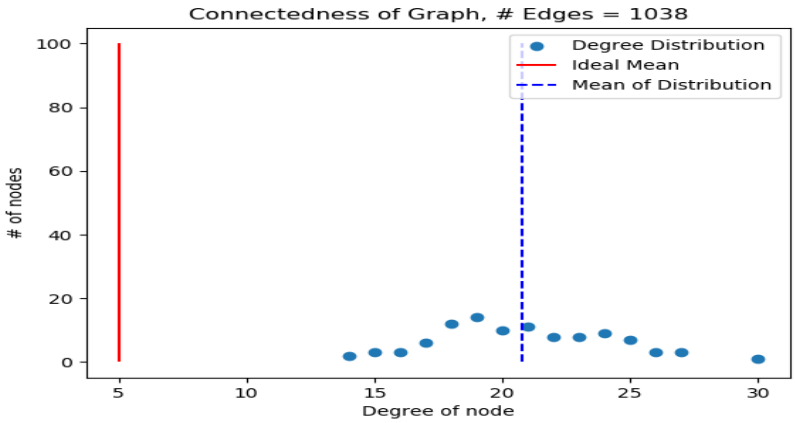

Fig. 5.2.12: Degree Distribution - Total Degree

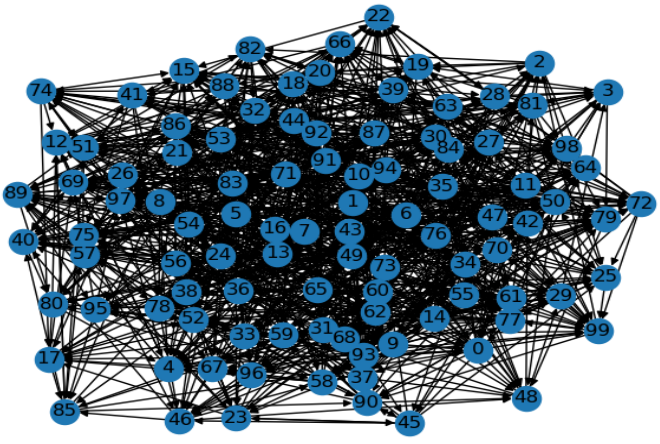

Fig. 5.2.13: Degree Distribution - Generated Graph

5.3 Implementation

In this section, we discuss the implementation details, and tools & technologies that backs the proposed method. This section gives a brief overview of how each aspect was implemented and tested. At the end of the section, we also compare our results with related methods to show where our proposed method stands.

5.3.1 Platform Technology

The code and the blockchain logic runs on the industry-standard containerization platform called docker. Docker is an open-source Platform As A Service (PAAS) technology that enables base-OS independent deployment bundles into isolated sub-systems called containers. In this case, every node in our blockchain network is a container. Because of the high level of abstraction provided by docker, this makes it an ideal candidate as the manageability of the system gets straight forward.

5.3.2 Blockchain Specifications & System Design

We design our blockchain that does simple currency transfer from one account to another. The block size is of 1KB and for every iteration, 4 non-conflicting transactions will be taken up from the pool and will be run over consensus. As part of the system data or actual database, to keep things lightweight, instead of using a heavy SQL schema DB engines, we use document store, were every registered user account will have an associated document which stores user information, like name, account number, and balance. A sample snapshot is as shown in Figure 5.3.1.


```
{
    "Ac_no" : "AC3456",
    "Name": "Andy",
    "Balance": 2050
}
```


Fig. 5.3.1: User Account

The file for AC3456 is a JSON document has the same file name as Ac_no. For this example, the name of the file will be AC3456.json.When a leader pickups the transaction from the transaction pool, it validates the existence of the account and then validates the balance of entities that has to be manipulated. As for the blockchain ledger that is generated as part of the blockchain, it is also a JSON file. The ledger and the data of the system are stored on the host machine and are mounted on the container in form of mount volumes. Ledger is a JSON file. The screenshot of the JSON file is shown in Figure 5.3.2.


```
{
    "Block_1": {
        "PreviousHash": "39897c27070a97c6c464a2c21899167958aec8b93bc1c4439c046c858d42676d",
        "BlockCreationTime": "2019-07-07 20:56:26.893705",
        "MerkelRoot": "81818253068c41f4c4b633347a1fd0b100077071b5fec39203f7e52eaff4022e",
        "TransactionID": "TX_TEST",
        "Transaction": {
            "t2": "AC1325 AC354 120",
            "t4": "AC4387 AC4574 211",
            "t3": "AC6511  AC4672 101",
            "t1": "AC1232 AC1092 109"
        }
    },
    "Genesis": {
        "PreviousHash": " ",
        "BlockCreationTime": " ",
        "MerkelRoot": "c9b2e8a8415fbd4c162db91e470651afc7521054c7dce491a11f77eebd11790f",
        "TransactionID": "TX_Genesis",
        "Transactions": {
            "t2": "AC0000 AC0000 0",
            "t4": "AC0000 AC0000 0",
            "t3": "AC0000 AC0000 0",
            "t1": "AC0000 AC0000 0"
        }
    }
}
```


Fig. 5.3.2: Blockchain Ledger

Implementation was done over the docker and as for the technology stack, we used python for the entire implementation. The docker containers ran lightweight Alpine Linux for the underlying specifications. And data was mounted on containers as the persistent volume which was stored at the host level. System specifications and deployment specifications are shown in table 5.3.1 and 5.3.2.

Host Operating System	Debian 7
Memory	24 GB (Buffered)
CPU	Intel Xeon W3250 @ 2.8GHz
Core and Threads	8 cores, 16 threads
Storage	512 GB
Number of Compute Nodes	3

Table 5.3.1: System Configuration - Test Bed

Container Image	Apline Linux
Installations	Python 3.7, grpc python, gnupg, protocol buffers v3
Node Storage	Persistent Named Volumes

Table 5.3.2: Node Configuration - Container Specification

Communication between the nodes in the system is carried by the gRPC protocol paradigm. Message passing in gRPC is backed by protocol buffers v3 which is a serialized data structure which is platform, and stack independent. Using this, we define our data and compile it using the gRPC compiler and use it for the variety of data streams. We use gRPC because it is a universal high-performance RPC framework that scales well with massively distributed systems and has a bidirectional secure streamed message passing interface

5.3.3 Results

In the given literature all the methods that we discussed, uses a variety of ways to tackle the problem of scalability. Due to this, there is a need for a common benchmark that brings the core of different systems at the same level and units. For comparing our methods, we use the methodology given by IBM in their whitepaper where they have specified performance metrics to compare the system design and working of

different blockchain systems. We use the following metrics for the whitepaper to evaluate our model for comparisons:

1. Transaction Throughput (T)

   Transaction throughput is the rate at which transaction are processed, validated and committed by the blockchain system. In transaction throughput only valid blocks are taken into consideration. This parameter is measured in unit transactions per second (tps).

2. Transaction Latency (l)

   Transaction Latency can be defined as the time interval between time transaction was submitted and time it was committed over the network is called transaction latency in the blockchain. This helps to quantify the readiness of data over the network after a valid transaction is submitted.

We compare our method with others for Transaction Latency & Throughput and Computational Complexity on Node. We choose Zyzzyva, FastBFT, and Gosig from advance byzantine fault tolerant systems because those are some of the optimal designs with regards to consensus design. In terms of algorithmic, we compare our method with the majority of BFTs and evaluate the standings.

We evaluate our throughput and latency up to the network size of 140 as the given literature compares their processes with the same size. We only compare latency for 100 nodes as most of the literature has the same default setting at 100 nodes. The experiment results are shown in Table 5.3.3 and Table 5.3.4. The values shown in the table are the average of 150 experiments. The throughput results are rounded to the nearest integer.

From Table 5.3.3, we can see that FastBFT outperforms all the other protocols because of the tree-based communication topology. Also when N increases, FastBFT does not show higher intervals of degradation in throughput. The proposed methodology holds the sub-optimal position in standings and also has similar low depredations levels when the network size increases. PBFT i.e. state of art methodology stands

last because of high communication complexity, essentially reducing the throughput of the system. Comparing the latency of transactions in Table 5.3.4 we observe similar trends as we saw in throughput and out method turns out to be sub-optimal in the list. Furthermore, we also evaluate the single node per block proposal complexity of the methods in Table 5.3.5. The proposed method turns out to be sub-optimal when compared with other methods. Where FastBFT and Zyzzyva have overly optimized communication topology which can easily break over arbitrary failures, over method guarantees a strongly connected network with a single node simply $O(itr \cdot ln(N))$ complexity. For Table 5.3.5, please note that N is used as the same standard notation i.e. the number of nodes in the network.

Methods	**N = 50**	**N = 100**	**N = 140**
FastBFT	980	971	953
Zyzzyva	992	856	813
Proposed Method	781	764	747
Gosig	653	607	547
PBFT	517	469	413

Table 5.3.3: Transaction Throughput Results (tps)

Methods	**Latency**
FastBFT	112.6
Zyzzyva	236.4
Proposed Method	267.7
Gosig	392.6
PBFT	646.8

Table 5.3.4: Transaction Latency Results (ms)

Methods	**Communication Complexity**
FastBFT	$O(b)$; where b is branching factor of node
Zyzzyva	$O(1)$; under speculative gracious execution
Proposed Method	$O(itr \cdot ln(N))$
Gosig	$O(N \cdot m)$; where m is a fan-out parameter
PBFT	$O(N^2)$

Table 5.3.5: Computational Complexity of Methods

CHAPTER 6

Conclusion & Future Work

6.1 Conclusion

In this thesis, we address the scaling problem that is prevailing in the blockchain systems. To constraint the domain, we only consider the consortium blockchain as our area as it has direct and useful applications and is a potential candidate to replace the existing computing and audit systems. In the initial chapters, we define the utility of blockchain and also discuss methods used in literature to scale systems. From this, we pick up a novel problem of collaborating multiple blockchains that share data and validate with each other into one system with a significant reduced communication overheads and a instead of existing monolithic design we propose a framework backed by state machine replication to extend finality to update states of the nodes without participating into actual consensus process. Also, we propose a new novel communication topology to the stack which cut downs communication overhead drastically and optimizes the overall system.

In the later sections, we do an in-depth analysis of our model by checking overall complexity, empirical analysis, effects on the system by tweaking core parameters and with the developed proof of concept, we also benchmark our model with existing approaches. Finally, we can say that our model stands at a respectable position as compared to other models we have in literature without overcompensating any underlying assumptions.

6.2 Future Work

We restrict our model in a consortium environment. This model can be further tested with public blockchains where overall system components are different from private blockchains. For our experiment test-bed, all the experiments were executed on local systems so the model can be further extended to cloud deployment and can be evaluated for other test metrics such as geographical latency, off-chained database repositories and many more. Another interesting area to go through can be testing random graph network topology over other BFT protocols and compare it with default topology they adhere to. Furthermore, because of domain restriction, we only consider our model against arbitrary failures. There is a future scope in incorporating adaptive adversaries scenarios as well.

REFERENCES

[1] Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Singh, G., Smith, K., Sorniotti, A., Stathakopoulou, C., Vukoli, M., Cocco, S. W., and Yellick, J. (2018). Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, pages 30:1–30:15, New York, NY, USA. ACM. event-place: Porto, Portugal.

[2] Becker, J., Breuker, D., Heide, T., Holler, J., Rauer, H. P., and Bhme, R. (2013). Can We Afford Integrity by Proof-of-Work? Scenarios Inspired by the Bitcoin Currency. In Bhme, R., editor, *The Economics of Information Security and Privacy*, pages 135–156. Springer, Berlin, Heidelberg.

[3] Bentov, I., Lee, C., Mizrahi, A., and Rosenfeld, M. (2014). Proof of Activity: Extending Bitcoin's Proof of Work via Proof of Stake [Extended Abstract]y. *ACM SIGMETRICS Performance Evaluation Review*, 42(3):34–37.

[4] Berger, C. and Reiser, H. P. (2018). Scaling Byzantine Consensus: A Broad Analysis. In *Proceedings of the 2Nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*, SERIAL'18, pages 13–18, New York, NY, USA. ACM. event-place: Rennes, France.

[5] Bhardwaj, S. and Kaushik, M. (2018). BlockchainTechnology to Drive the Future. In Satapathy, S. C., Bhateja, V., and Das, S., editors, *Smart Computing and In-*

formatics, Smart Innovation, Systems and Technologies, pages 263–271, Singapore. Springer.

[6] Bollobs, B. and Bla, B. (2001). *Random Graphs*. Cambridge University Press. Google-Books-ID: o9WecWgilzYC.

[7] Castro, M. and Liskov, B. (2002). Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461.

[8] Corso, A. (2019). Performance Analysis of Proof-of-Elapsed-Time (PoET) Consensus in the Sawtooth Blockchain Framework.

[9] Costan, V. and Devadas, S. (2016). Intel SGX Explained. *IACR Cryptology ePrint Archive*, 2016:86.

[10] Dannen, C. (2017). *Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners*. Apress, Berkeley, CA.

[11] Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., and Terry, D. (1987). Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, PODC '87, pages 1–12, Vancouver, British Columbia, Canada. Association for Computing Machinery.

[12] Distler, T., Cachin, C., and Kapitza, R. (2016). Resource-Efficient Byzantine Fault Tolerance. *IEEE Transactions on Computers*, 65(9):2807–2819.

[13] Eberhardt, J. and Heiss, J. (2018). Off-chaining Models and Approaches to Off-chain Computations. In *Proceedings of the 2Nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*, SERIAL'18, pages 7–12, New York, NY, USA. ACM. event-place: Rennes, France.

[14] Eberhardt, J. and Tai, S. (2017). On or Off the Blockchain? Insights on Off-Chaining Computation and Data. In De Paoli, F., Schulte, S., and Broch Johnsen,

E., editors, *Service-Oriented and Cloud Computing*, Lecture Notes in Computer Science, pages 3–15. Springer International Publishing.

[15] Erdos, P. and Alfred, R. (1959). On the evolution of random graphs.

[16] Gilad, Y., Hemo, R., Micali, S., Vlachos, G., and Zeldovich, N. (2017). Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 51–68, New York, NY, USA. ACM. event-place: Shanghai, China.

[17] Goldreich, O., Micali, S., and Wigderson, A. (1991). Proofs That Yield Nothing but Their Validity or All Languages in NP Have Zero-knowledge Proof Systems. *J. ACM*, 38(3):690–728.

[18] Gupta, I., Kermarrec, A. M., and Ganesh, A. J. (2002). Efficient epidemic-style protocols for reliable and scalable multicast. In *21st IEEE Symposium on Reliable Distributed Systems, 2002. Proceedings.*, pages 180–189.

[19] Hopwood, D., Bowe, S., Hornby, T., and Wilcox, N. (2016). Zcash protocol specification. *Tech. rep. 20161.10. Zerocoin Electric Coin Company, Tech. Rep.*

[20] Kapitza, R., Behl, J., Cachin, C., Distler, T., Kuhnle, S., Mohammadi, S. V., Schrder-Preikschat, W., and Stengel, K. (2012). CheapBFT: resource-efficient byzantine fault tolerance. In *Proceedings of the 7th ACM european conference on Computer Systems*, EuroSys '12, pages 295–308, Bern, Switzerland. Association for Computing Machinery.

[21] Kermarrec, A.-M. and van Steen, M. (2007). Gossiping in Distributed Systems. *SIGOPS Oper. Syst. Rev.*, 41(5):2–7.

[22] Kotla, R., Alvisi, L., Dahlin, M., Clement, A., and Wong, E. (2007). Zyzzyva: speculative byzantine fault tolerance. *ACM SIGOPS Operating Systems Review*, 41(6):45–58.

[23] Kowalczyk, W. and Vlassis, N. (2005). Newscast EM. In Saul, L. K., Weiss, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 17*, pages 713–720. MIT Press.

[24] Li, P., Wang, G., Chen, X., and Xu, W. (2018). Gosig: Scalable Byzantine Consensus on Adversarial Wide Area Network for Blockchains. *arXiv:1802.01315 [cs]*. arXiv: 1802.01315.

[25] Li, W., Sforzin, A., Fedorov, S., and Karame, G. O. (2017). Towards Scalable and Private Industrial Blockchains. In *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, BCC '17, pages 9–14, New York, NY, USA. ACM. event-place: Abu Dhabi, United Arab Emirates.

[26] Liu, J., Li, W., Karame, G. O., and Asokan, N. (2019). Scalable Byzantine Consensus via Hardware-Assisted Secret Sharing. *IEEE Transactions on Computers*, 68(1):139–151.

[27] Maxwell, G., Poelstra, A., Seurin, Y., and Wuille, P. (2019). Simple Schnorr multi-signatures with applications to Bitcoin. *Designs, Codes and Cryptography*, 87(9):2139–2164.

[28] Micali, S., Rabin, M., and Vadhan, S. (1999). Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, pages 120–130. ISSN: 0272-5428.

[29] Molloy, M. and Reed, B. (1995). A critical point for random graphs with a given degree sequence. *Random Structures & Algorithms*, 6(2-3):161–180.

[30] Nakamoto, S. (2019). Bitcoin: A Peer-to-Peer Electronic Cash System. Technical report, Manubot.

[31] Sabt, M., Achemlal, M., and Bouabdallah, A. (2015). Trusted Execution Environment: What It is, and What It is Not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 57–64. ISSN: null.

[32] Saleh, F. (2020). Blockchain Without Waste: Proof-of-Stake. SSRN Scholarly Paper ID 3183935, Social Science Research Network, Rochester, NY.

[33] Schneider, F. B. (1993). *Chapter 7: Replication Management Using the State-machine Approach, Distributed Systems, 2nd edn.*

[34] Syta, E., Tamas, I., Visher, D., Wolinsky, D. I., Jovanovic, P., Gasser, L., Gailly, N., Khoffi, I., and Ford, B. (2016). Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 526–545.

[35] Szydlo, M. (2004). Merkle Tree Traversal in Log Space and Time. In Cachin, C. and Camenisch, J. L., editors, *Advances in Cryptology - EUROCRYPT 2004*, Lecture Notes in Computer Science, pages 541–554. Springer Berlin Heidelberg.

[36] Wahby, R. S., Tzialla, I., Shelat, A., Thaler, J., and Walfish, M. (2018). Doubly-Efficient zkSNARKs Without Trusted Setup. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 926–943. ISSN: 2375-1207.

[37] Yang, T.-M. and Maxwell, T. A. (2011). Information-sharing in public organizations: A literature review of interpersonal, intra-organizational and inter-organizational success factors. *Government Information Quarterly*, 28(2):164–175.

[38] Yang, Y. (2018). LinBFT: Linear-Communication Byzantine Fault Tolerance for Public Blockchains. *arXiv:1807.01829 [cs]*. arXiv: 1807.01829.

[39] Zhang, K. and Jacobsen, H.-A. (2018). Towards Dependable, Scalable, and Pervasive Distributed Ledgers with Blockchains. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1337–1346, Vienna. IEEE.

[40] Zhong, M. (2002). A faster single-term divisible electronic cash: ZCash. *Electronic Commerce Research and Applications*, 1(3):331–338.

VITA AUCTORIS

NAME:	Parth Anand Shukla
PLACE OF BIRTH:	Vadodara, Gujarat, India
YEAR OF BIRTH:	1997
EDUCATION:	Higher Secondary Diploma – Science Stream, Atharva Vidyalaya, Vadodara, Gujarat, India, 2014
	Charotar University of Science and Technology (CHARUSAT), B.Tech in Computer Engineering, Anand, Gujarat, India, 2018
	University of Windsor, M.Sc in Computer Science, Windsor, Ontario, Canada, 2020