Enhancing Blockchain Performance and Security: Pushing the Limits of Decentralized Applications

Deepal Nalindra Tennakoon

A thesis submitted to fulfil requirements for the degree of

Doctor of Philosophy

School of Computer Science

Faculty of Engineering

The University of Sydney

November 2023

Statement of Originality

This is to certify that to the best of my knowledge, the content of this thesis is my own work. This thesis has not been submitted for any degree or other purposes.

I certify that the intellectual content of this thesis is the product of my own work and that all the assistance received in preparing this thesis and sources have been acknowledged.

Deepal Nalindra Tennakoon
August 28, 2023

Authorship Attribution

The results presented in this dissertation were published in the 5 publications (includes two core A ranked publications) mentioned below and can be found in relevant chapters as outlined below:

(1) Chapter 2 contains information from the materials published in [1, 2, 3, 4]. I am the primary author of these publications who developed the concepts, proved the concepts, implemented the concepts, evaluated the concepts, and compared the developed solution against state-of-the-art using empirical and non-empirical methods.

[1] Deepal Tennakoon, Yiding Hua, and Vincent Gramoli. "Smart Redbelly Blockchain: Reducing Congestion for Web3". In: *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2023, pp. 940–950. DOI:10.1109/IPDPS54959.2023.00098.

[2] Deepal Tennakoon and Vincent Gramoli. "Blockchain Proportional Governance Reconfiguration: Mitigating a Governance Oligarchy". In: *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 2023, pp. 545–556. DOI: 10.1109/CCGrid57682.2023.00057.

[3] Deepal Tennakoon and Vincent Gramoli. "Dynamic blockchain sharding". In: *5th International Symposium on Foundations and Applications of Blockchain 2022 (FAB 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2022.

[4] Deepal Tennakoon and Vincent Gramoli. "Transparent Sharding". In: *Data Engineering*(2022), p. 37.

(2) Chapter 3 contains material published in [1]. I am the primary author of this manuscript. My contributions included, developing a solution to bridge the gap in literature, proving the solution, implementing the solution and evaluating the solution against state-of-the-art.

[1] Deepal Tennakoon, Yiding Hua, and Vincent Gramoli. "Smart Redbelly Blockchain: Reducing Congestion for Web3". In: *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2023, pp. 940–950. DOI:10.1109/IPDPS54959.2023.00098.

(3) Chapter 4 contains material published in [3] and [4]. I am the primary author of these manuscripts. I developed, implemented and evaluated the protocol presented in [3]. I contributed towards writing [4].

[3] Deepal Tennakoon and Vincent Gramoli. "Dynamic blockchain sharding". In: *5th International Symposium on Foundations and Applications of Blockchain 2022 (FAB 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2022.

[4] Deepal Tennakoon and Vincent Gramoli. "Transparent Sharding". In: *Data Engineering*(2022), p. 37.

(4) Chapter 5 contains material under submission and available as a preprint [5]. I am the primary author of [5]. My tasks included implementing and evaluating the presented solution, and writing the manuscript.

[5] Deepal Tennakoon, Yiding Hua, and Vincent Gramoli. "CollaChain: A BFT collaborative middleware for decentralized applications". In: *arXiv preprint arXiv:2203.12323* (2022).

(5) Chapter 6 contains material published in [2]. I am the primary author of [2]. My contributions ranged from developing, implementing and evaluating the solution as well as writing part of the manuscript.

[2] Deepal Tennakoon and Vincent Gramoli. "Blockchain Proportional Governance Reconfiguration: Mitigating a Governance Oligarchy". In: *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 2023, pp. 545–556. DOI: 10.1109/CCGrid57682.2023.00057.

In addition to the statements above, in cases where I am not the corresponding author of a published article, permission to include the published material has been granted by the corresponding author(s).

Deepal Nalindra Tennakoon
Date: August 28, 2023

As the primary supervisor for the candidature upon which this thesis is based, I confirm that the authorship attribution statements above are correct.

Associate Prof. Vincent Charles Gramoli
Date: August 28, 2023

This thesis is dedicated to my wife and parents, whose immense love has kept me going during tough times.

Acknowledgements

The journey of a Ph.D. student is typically filled with ups and downs. This was particularly true in my case due to personal circumstances. It certainly has been an emotional roller coaster for me. I have been through it all from being in lockdown alone to seeing my father go through heart surgery. Sometimes we all wonder whether it is worth doing a Ph.D. to achieve a research result that may at best be a modest contribution to the grand history of groundbreaking research. Now that I have reached the end of my Ph.D. journey, I feel very differently. A Ph.D. is not all about achieving extraordinary research results or publishing manuscripts, it is about learning how to be a researcher.

Throughout these 3.5 years there have been numerous people who have helped, encouraged and believed in me. I would like to take this opportunity to thank you all. I am extremely grateful. I would not be here if not for all your help and support.

Firstly, I would like to thank my supervisor Vincent for taking me under his wing and giving me the opportunity to explore blockchain research. I am extremely grateful for your patience, understanding, and most importantly for believing in me. You have been like a father to me throughout this journey and have always been someone I can count on.

I would like to thank my family for being my strength during these hard times. Especially my wife (Umodhi), parents (Sudath and Nadheera), and brother (Shamal). Since our marriage in 2021, my wife moved to Sydney and made several sacrifices for my success. Thank you for being understanding and for your unconditional love and support. I would also like to thank my parents for loving me unconditionally and encouraging me to pursue a Ph.D. Since my childhood you have always emphasized the importance of education, and have always been behind me, supporting me all the way. I am who I am because of you and I love you both very much. I hope that I have made you proud. My little brother was my playmate when we were kids. When our parents were not around we used to keep each other company and look after each other. I hope my journey encourages you to do a Ph.D. as well.

Next, I would like to thank my dear friends from the Concurrent Systems Research Group (CSRG) at the University of Sydney, including Chris, Alejandro, Gauthier, Pouriya, Pierre, and Andrei. I would like to thank especially Chris. Chris, I always looked up to you and what you have achieved. You have helped review my research manuscripts as well as my thesis. The advice you have given me on life has been invaluable. I will miss our Signal chats. Alejandro, you have also been a close friend during my PhD journey. You have been someone I could share my sorrows and challenges. Gauthier, you have been an amazing Postdoc who always brought out the best in all of us at CSRG. I will always remember your drinks treat for my birthday. Pouriya, you are an amazing Ph.D. candidate. I will always remember our hikes and discussions. Thank you for being a great friend.

I would like to thank my Master's supervisor Rolando at Deakin for introducing me to security research, and all my lecturers and tutors in my bachelor's and master's degree including Feifei Chen, Iman Avazpour, Morshed Chowdhury, Asitha Bandaranayake, Roshan Ragel and Dhammika Elkaduwa. Finally, I would like to thank all my schoolmates at Trinity College, Kandy. Although I would not name everyone here as there are so many of you, just know that

you lot have been my closest friends. I apologize profusely to those who I forgot to mention. Thank you for everything you have done for me.

Abstract

Decentralized Applications (DApps) have seen exponential growth in the past decade leading to a new paradigm known as Web3. Web3 is the ecosystem formed by the execution of multiple DApps. Blockchains offer a platform for DApp executions. However, the performance and security of current blockchains is limited and impair the adoption of Web3. More specifically, for demanding DApp workloads, modern blockchains perform poorly or lose transactions.

This thesis presents various contributions to enhance blockchain performance and security to widen the adoption of Web3. To enhance blockchain performance for DApp executions, we first present the Smart Redbelly Blockchain (SRBB). SRBB enhances DApp performance by reducing blockchain congestion. SRBB alone is not sufficient to service multiple demanding DApp workloads. Therefore, we introduce a DApp-oriented dynamic transparent sharding mechanism that concurrently execute DApps in separate shards. To boost the DApp performance of SRBB, we present a decoupled variant of SRBB known as Collachain.

While blockchain performance is critical, existing blockchain designs are vulnerable to the formation of an oligarchy in the governance that can dictate the outcome of the protocol. Such an oligarchy can lead to the insecure execution of DApps, impairing the adoption of Web3. To mitigate the formation of an oligarchy in blockchain governance, we finally present a proportional governance protocol that proportionally elects a diverse set of governors to mitigate an oligarchy in the governance process.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A blockchain has often been regarded as a complex and convoluted distributed system that executes transactions amongst unknown sets of peers. However, it has often been overlooked that the fundamental principles forming a blockchain have been studied in distributed systems research for many years [6]. Put simply, a blockchain exhibits the properties of a fully distributed State Machine Replication (SMR) protocol [7]. Thus, a blockchain on a high level can be described as a set of machines executing user requests in an agreed order to maintain a global identical state. To agree on the execution order, machines communicate with each other to propose and vote on batches of user requests batched in a particular order (i.e. blocks). Machines agree to execute a block once the block receives a majority of votes, a process known as distributed consensus. The distinction between SMRs and blockchains is that blockchains require an agreed block from a consensus instance to be related to the block decided in the previous consensus instance. More specifically, blockchains, after agreeing upon a block and executing the said block, adds the block to an append-only ledger where each block points to its predecessor, essentially forming a chain of blocks. In contrast, SMRs concatenate outputs of consensus instances without relating the output of a consensus instance to its predecessor [8]. While consensus is trivial if all machines adhere to protocol, disagreement can occur if faulty or corrupt machines exist. To reach agreement despite faulty machines, Crash Fault Tolerant (CFT) consensus algorithms were implemented on blockchains [9]. To ensure agreement in the presence of arbitrarily failing machines, Byzantine Fault Tolerant (BFT) consensus algorithms were used [10, 11, 12].

Since its inception as a decentralized payment system [13], blockchain has seen widespread use over the years. The Blockchain industry is predicted to be worth 403 Billion USD by 2030 [14]. Today there are over 8800 active cryptocurrencies powered by blockchains and 10% of the world's population own cryptocurrencies [15]. In 2014, Ethereum [16] introduced the ability to interact with code and execute functions on the blockchain through transactions. This gave rise to applications that executed on the blockchain which were dubbed Decentralized Applications (DApps). More specifically, a DApp consists of an interface (e.g., web front-end) that communicates via Remote Procedure Calls (RPC) with a back-end piece of code executing on the blockchain. The execution of the DApp back-end code on the blockchain makes these applications inherently decentralized as there is no single authority controlling the execution

of blockchains [17]. The recent growth of blockchain users has been largely driven by the growth of DApps. The number of unique users using DApps increased 7× in 2021 [18]. A majority of these DApps included Decentralized Finance (DeFi) applications (e.g., Decentralized Exchanges), decentralized games (e.g., Alien Worlds), and Non-fungible Tokens (NFTs) (e.g., Bored Ape [19]) executed on blockchains [20].

The ecosystem that is formed by multiple DApps executing on multiple blockchains is known as Web3 [21, 1]. Web3 is the latest iteration of the web providing decentralization over centralized web applications controlled by big technology companies (e.g., Google, Facebook, and Amazon). Centralized web applications are notorious for censorship and data manipulations [22]. For instance, the processes followed when Facebook removes user content that violates its community standards lack transparency [23]. News feeds on Google and Facebook show biased content towards certain political beliefs [24]. Moreover, the business model of these big technology companies involves tailoring advertisements to users based on the personal information collected, which is a form of data manipulation. Web3 is an ideal replacement for centralized web applications that manipulate data and censor content [25]. However, to widen the use of Web3, one first needs to solve the existing performance and security issues that plague modern blockchains [1, 25, 2, 3]. More specifically, modern blockchains suffer from performance degradation and transaction losses induced by congestion from demanding DApp workloads [26, 27, 28, 1]. These blockchains are also prone to the formation of an oligarchy, potentially leading to blockchain attacks such as double-spending [2].

Recent blockchain performance improvement solutions focus on two main areas: (1) Improvements to the blockchain itself (*Layer 1*, or blockchain layer) through protocol and architectural changes, and (2) offloading request executions and state storage on the blockchain to a separate dedicated environment off-chain (*Layer 2*) while still utilizing the security of the blockchain layer [29]. Layer 1 blockchain performance enhancing solutions include amongst other things blockchain sharding, decoupling blockchain components, and concurrent request executions [3, 30, 31, 32, 33, 34]. The most popular Layer 2 solutions are rollups, both optimistic and Zero-Knowledge (ZK) [29, 35]. Rollups consist of two main steps: (a) transactions are executed off-chain producing proofs of execution, (b) these proofs are submitted to the Layer 1 blockchain where they are verified and, if correct and unopposed, will be added to the blockchain through the consensus. The verification of execution proofs on the blockchain does not require the blockchain to execute the request or have any knowledge of the execution parameters, hence the verification essentially requires zero knowledge.

In this thesis, we focus on improving the performance and security of the blockchain layer to support the execution of an ever-growing ecosystem of DApps and contribute towards the wider adoption of Web3. To this end, we present various novel contributions to improve the performance and security of the blockchain protocol. As Layer 2 solutions are built on top of the blockchain layer, the performance and security improvements we present on the blockchain layer in this body of work can be relied upon by Layer 2 solutions to further improve blockchain performance.

In summary, this thesis presents various enhancements in blockchain performance and secu-

rity for DApp executions with the motivation of widening the adoption of Web3.

Motivation: Widening the adoption of Web3

Goal: Enhance blockchain performance and security for DApp executions

Based on our thesis goal, we list below our two main objectives.

1.1 Objectives

Objective 1: Enhance blockchain performance for DApp executions: Over the years various proposals were introduced to enhance blockchain performance by changing the blockchain protocol [3, 30, 31, 32, 33, 34, 10, 36]. Some of these proposals focused on enhancements in the blockchain consensus [36, 32], while others focused on improving transaction executions and storage of blockchains [34, 32, 37]. There was yet another category of approaches that improved blockchain performance by changing the architecture of blockchains [32, 33, 5]. Some of these enhancements weaken security assumptions to achieve performance [32, 33]. Many secure modern blockchains built upon recent improvements in the blockchain protocol are not able to execute real DApp workloads without transaction losses and a severe decline in performance compared to their reported performances [37, 10, 38, 16, 11, 39]. Thus, there is a need to enhance the performance of the blockchain protocol to support the execution of real DApp workloads.

As a result, our first objective of enhancing blockchain performance encapsulates the sub-objectives below that enable the blockchain protocol to support real DApp workloads.

- **Objective 1.1:** *Investigate the reasons why modern blockchains cannot support real DApp workloads.*
- **Objective 1.2:** *Design, develop and evaluate a secure blockchain capable of supporting real DApp workloads.*
- **Objective 1.3:** *Design, develop and evaluate a blockchain sharding approach that improves blockchain performance by executing DApps concurrently.*
- **Objective 1.4:** *Design, develop and evaluate a decoupled variant of the initially developed blockchain to further improve DApp performance.*

Improving DApp performance through various methods helps towards widening the adoption of Web3.

Objective 2: Enhance blockchain security for DApp executions: Blockchain governance is the process followed to make decisions and modify the blockchain protocol [2]. Thus, governance encapsulates a wide range of tasks that also includes blockchain consensus. To achieve scalability in an open network modern blockchains restrict the number of machines executing governance [2, 10]. To allow any user to execute governance, many blockchains offer a periodic rotation of the set of machines that govern [10, 30, 31, 2]. However, the governance protocols used in modern blockchains to elect governors are vulnerable to the formation of a governance oligarchy that can lead to double-spending attacks [2]. Thus, our objective of enhancing blockchain security encapsulates the sub-objectives below that mitigate a governance oligarchy in blockchains and facilitate the secure execution of DApps.

- **Objective 2.1:** *Design and develop a blockchain governance protocol to mitigate a governance oligarchy.*
- **Objective 2.2:** *Prove the properties that the governance protocol provides.*
- **Objective 2.3:** *Evaluate the developed governance protocol on blockchains to observe its feasibility.*

Providing means to improve the security of DApp execution through a governance protocol that mitigates a governance oligarchy contributes towards widening the adoption of web3.

1.2 Contributions

This dissertation presents four novel contributions aligning with the research objectives presented previously and our goal of enhancing blockchain performance and security for DApp executions. First, we identify bottlenecks in modern blockchains that make them incapable of supporting real DApp workloads and then we present a highly performing blockchain that supports real DApp workloads. Second, we present a novel dynamic sharding solution for blockchains that supports the dynamically changing user request rates of DApps and enables the concurrent execution of DApps. Third, we present a decoupled variant of the initially presented highly performing blockchain to further improve DApp performance. Finally, we present a novel blockchain proportional governance protocol to mitigate a governance oligarchy in blockchains to ensure the secure execution of DApps.

In summary, our thesis presents the following contributions:

1. The Smart Redbelly Blockchain (Objective 1.1 & 1.2) - We identify *blockchain congestion* (the saturation of transaction queues in the blockchain) to be the reason that modern blockchains cannot support real DApp workloads [28]. To reduce blockchain congestion and support real DApp workloads, we develop a provably secure permissionless blockchain known as Smart Redbelly Blockchain (SRBB). SRBB uses the DBFT consensus and Redbelly blockchain's superblock optimization. Unlike the Redbelly blockchain, SRBB (1)

supports smart contract and DApp executions (2) features a transaction validation reduction (TVPR) to reduce blockchain congestion and (3) features a reward-penalty mechanism (RPM) to reduce blockchain congestion in the presence of corrupt machines. Our results with 200 machines spanning 5 continents and 10 regions demonstrate that SRBB outperforms 6 modern blockchains: Algorand, Diem, Avalanche, Solana, Ethereum, and Quorum [2] for real DApp workloads [28]. Unlike other evaluated blockchains, SRBB is able to commit 100% of the transactions for the demanding NASDAQ and Uber DApp workloads.

2. DApp-oriented Dynamic Transparent Blockchain Sharding (Objective 1.3) - To concurrently execute multiple DApps to further improve performance and to support the dynamically changing demand of DApps, we present a DApp-oriented dynamic transparent blockchain sharding protocol. In our sharding protocol, each DApp or a group of related DApps executes on at most one shard. This allows the concurrent execution of DApps without cross-shard transactions, a concept known as DApp-oriented or service-oriented sharding [40]. Our sharding protocol is also able to change the number of shards and the size of a shard at runtime while also allowing users to transparently query the sharding configuration. The sharding protocol we present can be adapted to any smart contract supported blockchain but for evaluation purposes, we build our sharding protocol on SRBB. With just 3 shards of SRBB, each executing the NASDAQ, Uber and FIFA DApp workloads concurrently, we achieve an average throughput of 2828.87 TPS and a peak throughput of 9523 TPS.

3. Collachain: Decoupling Smart Redbelly Blockchain (Objective 1.4) - To further extend our objective of improving blockchain performance, we decouple the consensus and execution of SRBB to produce Collachain. We discuss the trade-offs the decoupling entails and show that Collachain yields a 33% increase in peak throughput compared to SRBB.

4. Blockchain Proportional Governance: Mitigating a Governance Oligarchy (Objective 2) - To enhance blockchain security, we present a novel blockchain proportional governance protocol that relies on a proportional election protocol to mitigate an oligarchy amongst the governors. We prove the properties of our solution. Finally, to present the feasibility of our proportional governance mechanism, we implement and evaluate it on SRBB and Ethereum Proof-of-Authority (PoA) which are two blockchains on the slower and faster end of the blockchain performance spectrum. In a geo-distributed evaluation spanning 200 nodes, we demonstrate that our solution can elect 200 governors from 500 candidates with 1000 voters within 6-12 minutes.

1.3 Thesis Outline

Chapter 2 presents the background. It explains blockchain concepts in detail, and related work, which forms the foundation of concepts this thesis builds upon.

Chapter 3 presents the main contribution of this thesis which is the Smart Redbelly Blockchain. SRBB enhances blockchain performance for DApp executions. More specifically, SRBB is a provably secure blockchain able to support real DApp workloads [1, 28].

Chapter 4 introduces a DApp-oriented dynamic transparent sharding protocol that enhances blockchain performance by concurrently executing DApps in separate shards. The sharding protocol we present is dynamic in that it provides the ability to adjust shards according to DApp demand at runtime. Our sharding protocol is also transparent allowing anyone to query the blockchain and identify the sharding configuration.

Chapter 5 builds upon SRBB presented in Chapter 3 to produce a decoupled blockchain, Collachain, to further enhance blockchain performance of SRBB.

Chapter 6 presents a blockchain proportional governance protocol to enhance blockchain security. This proportional governance mechanism is designed to mitigate a governance oligarchy.

Chapter 7 concludes by presenting how we fulfill the goals and objectives identified in the thesis. This chapter also discusses future research directions that include widening the evaluations and combining the four individual contributions to produce a single overarching system.

Chapter 2

Background

In this chapter, we present blockchain concepts and the related work to the content presented in the subsequent chapters. First, we present blockchain preliminaries that define the core components of a blockchain. Next, we define blockchain network models that most common blockchains function upon. The blockchain consensus problem is defined subsequently as our proofs in subsequent chapters rely on this definition. As the motivation of our thesis is to widen the adoption of Web3, we introduce Web3 technologies next. To provide context to Chapter 3, we then describe blockchain performance metrics and evaluation tools. Subsequently, we present blockchain governance concepts to form the background for Chapter 6. Blockchain sharding concepts presented next provide background knowledge for Chapter 4. Finally, we conclude by presenting several blockchain consensus protocols and blockchain attacks.

2.1 Blockchain Preliminaries

While variations exist in blockchain implementations, there are some common concepts found in many modern blockchains. We now introduce these concepts that form a blockchain.

2.1.1 Blockchain nodes

A blockchain consists of nodes which are machines such as PCs, laptops, or other pieces of hardware connected together over a network.

Clients are blockchain nodes that send read and write requests to the blockchain. Note that the term *client* is used to define implementations of Ethereum (e.g., Geth client – Ethereum's Golang implementation) by the Ethereum community but we identify a *client* solely as a sender of requests to the blockchain.

Validators, also known as miners in some blockchain implementations, are blockchain nodes that perform a number of critical tasks. Mainly, validators (1) validate client write requests (2) propose validated client write requests in batches known as blocks to the network of validators (3) decide the execution order of client write requests (4) execute write requests in total order, and (5) service client read requests. Depending on the blockchain, validators also perform other tasks such as storing the client write requests after execution for auditability and transparency.

Note that validators can also behave as clients sending write or read requests to the blockchain. Throughout this thesis, we mention two types of validators. Namely, *Correct* and *Byzantine* validators.

Validators that follow the blockchain protocol are known as *correct* validators. For example, a correct validator does not: (1) propose blocks with invalid transactions, (2) equivocate when reaching agreement on the transaction order, and (3) censor transactions. Validators that deviate from the blockchain protocol either arbitrarily or otherwise are known as *Byzantine* validators. The term *Byzantine* stems from the Byzantine Generals problem [6], where a group of generals from the Byzantine army can only successfully conquer an enemy fortress if a majority of generals decide to attack. It follows that if the total number of generals is n, and those that send arbitrary messages to confuse the generals' decision is f, then deviating generals should be less than a third of the total number of generals to ensure all correct generals decide to attack ($n/3 > f$). However, in the blockchain context, Byzantine validators, in addition to sending arbitrary messages to cause disagreement on the transaction order, can also perform other protocol-deviating behaviors such as proposing invalid transactions to blocks, censoring transactions, sending erroneous messages, or delaying messages.

2.1.2 Accounts/Wallets

An account enables a node to send write requests to the blockchain. An account mainly contains an address that uniquely identifies the account. The account balance indicates the amount of funds contained in the account. This notion of a blockchain account is analogous to a bank account in that a bank has an account number and a balance whereas a blockchain account has an address and a balance. Thus, a blockchain account is usually called a *wallet*.

There are a number of differences between a bank account and a blockchain account (i.e., a wallet). Firstly, a wallet is uniquely identified from its address, which is derived from the public key of an asymmetric key pair generated upon the creation of the wallet. Secondly, the wallet is secured by the cryptography of the private-public key pair. In other words, one can only send requests from the wallet if they possess the private key of the wallet. This is because requests sent from a wallet require a valid signature from that wallet's private key. Blockchain nodes only process write requests upon verifying the signature of the sent request with the corresponding public key of the sending wallet and identifying that the request indeed came from the owner of the private key of the wallet. Thirdly, in most blockchains [16, 1, 11], a wallet has a sequence number that increments every time the wallet owner sends write requests to the blockchain. This sequence number is also known as a *nonce* in some blockchains (e.g., Ethereum [16]), and helps a blockchain execute requests in order. More specifically, if two requests are sent from a wallet, the request with the lower sequence number should be executed prior to the request with the higher sequence number.

2.1.3 Transactions

Transactions are write requests sent by clients to the blockchain. In modern blockchains, transactions can be of three main types: native payments that transfer funds between accounts/wal-

lets, code deployments that upload code to be executed by the blockchain, and code executions that invoke functions in the uploaded code.

A transaction in blockchains has a specific structure. Moreover, a transaction commonly consists of (1) the sender wallet address which is the address of the account sending the transaction, (2) the receiver wallet address which is the wallet receiving funds, (3) the sequence number (i.e., nonce), and (4) amount of funds transferred if the transaction is a native payment transaction, the byte code of the code to be uploaded if the transaction is a code deployment or the byte code of the function invocation if the transaction is a code execution. Two transactions can conflict if they read the same data and at least one transaction is a write request [1, 41].

2.1.4 State Machine (SM)

The SM is a virtual machine with an instruction set, also known as a stack machine, that is able to execute transactions, maintain the blockchain state, and store executed transactions. When a transaction is executed, the SM updates the state referred to in the transaction (i.e., sender wallet balance, receiver wallet balance, code state).

The *Ethereum Virtual Machine (EVM)* is the SM of the Ethereum blockchain. Its different implementations and various optimized versions are used in a number of blockchains [16, 11, 1, 42, 43, 44]. The EVM stores the blockchain state in a specialized tree known as the Merkel Patricia Tree (MPT). Whenever a transaction is executed, the EVM updates the MPT. The nodes of the MPT are stored in memory and branches of this tree are periodically flushed to the disk when the size of the MPT exceeds a certain threshold. Keeping the MPT in memory allows the blockchain state to be accessed quickly by clients.

2.1.5 Gas

Gas is a metric used in Ethereum [16] and other blockchains that use the Ethereum Virtual Machine [1, 43] to measure the computational complexity required to execute specific operations in a blockchain. The execution of transactions requires executing operations on the blockchain that expends computational resources. Validators are compensated for expending computation resources with a fee based on the gas value. A unit of gas has a monetary value known as the gas price based on the usage of the blockchain network, which allows the amount of gas a transaction uses to translate to a transaction fee in cryptocurrency. When submitting a transaction, a client should include a reasonable gas value within the transaction known as the gas limit. The gas limit represents the maximum amount of gas a client is willing to spend on a transaction. If a validator executes a transaction that exceeds the gas limit (the transaction runs out of gas), the transaction is reverted and the amount of gas spent is charged as a fee from the sender's wallet address.

2.1.6 Blocks

A block is simply a batch of transactions. The contents of a block change based on the blockchain implementation. However, a block typically contains a set of transactions, a hash representing

the contents of the block, the hash of the previous block, also known as the parent block, which points the block to its predecessor, and an index, also known as the block number, which represents the location of the block in the chain of blocks. The first block in a blockchain is known as the *Genesis* block, which defines the initial state once a blockchain bootstraps. Unlike other blocks, the Genesis block does not point to a preceding block. When validators agree on the location of a block in the chain, a block is considered final or decided. Blockchains execute transactions within decided blocks triggering global state updates.

2.1.7 Transaction Pool

Transaction pool and mempool are two terms used to define the location at which transactions are stored temporarily soon after being received from a client, and before being included in a block. In most cases, the transaction pool resides on each validator node [16, 1]. A validator node creates blocks from transactions in its transaction pool and propagates these blocks to other validators.

2.1.8 Transaction Models

There are typically two main transaction models used in blockchains. The Unspent Transaction Output (UTXO) model used in blockchains such as Bitcoin [13] and Redbelly [12] uses transaction outputs from previous transactions as inputs to spend on new transactions. More specifically, if a user A wants to send an amount in cryptocurrency to user B, user A should use its unspent transactions as input and generate a new transaction which when executed will provide an unspent transaction to B. The amount unspent in the input of this transaction after execution will be credited back to A as a new unspent transaction after a proportion of it is deducted as the transaction fee. A can then use the unspent transaction it receives in subsequent transactions.

In contrast, the account/balance model used in Ethereum [16] and a number of other blockchains [1, 11, 45, 10] uses a transaction model similar to conventional bank transactions [45]. If a user A wants to transfer an amount in cryptocurrency to user B, A should have sufficient balance in their account to pay the transaction fee and the amount specified in the transaction. Once a transaction is executed, the sender's account balance and the receiver's account balance are updated according to the transferred amount. As the account/balance model reflects the incremental update of states and arbitrary amounts in cryptocurrency can be processed with a single transaction, it is more suited to implement complex application logic like smart contracts [45].

2.1.9 Blockchain

A blockchain is a decentralized and distributed system with multiple validator nodes communicating over a peer-to-peer network that (1) agrees upon the execution order of transactions sent by clients (2) executes agreed transactions, and (3) stores executed transactions in blocks, where each block points to its predecessor forming a chain of blocks also known as a *Blockchain*.

There are three main types of blockchains. Namely, permissionless, open, and permissionless blockchains. Permissionless blockchains do not place any restrictions on nodes to join or leave the network. Anyone can become a client or a validator. These blockchains are open to the public and are also known as public permissionless blockchains. The two largest blockchains Bitcoin [13] and Ethereum [16] are both public permissionless blockchains.

Open Blockchains allow any node to join or leave the network. However, specific validator tasks such as proposing blocks or agreeing on the order of transactions are restricted to a fixed set of nodes. An open blockchain can be made permissionless by periodically granting validator rights to nodes, allowing anyone in the network to become a validator. This is the process employed by Algorand [10].

Permissioned Blockchains are the opposite of permissionless blockchains and consist of an access control layer. Any node that receives permission to access the network can join the network. However, permission is usually provided to an exclusive set of nodes. In contrast to permissionless blockchains, these blockchains work with a limited number of nodes. A few notable examples of permissioned blockchains include Hyperledger [33] and Quorum [11].

2.1.10 Transaction Validation

Transaction validation is the task of checking whether the data encapsulated in a transaction conforms with the blockchain protocol. More specifically, transaction validation involves a number of sanity checks on a transaction. In modern blockchains, these transaction validations are performed twice [1]:

- **Eager validation**: Eager validation occurs when a validator receives a transaction either from another validator or a client. A validator then performs the following checks on the transaction:

  1. Is the transactions properly signed?
  2. Does the sender account have sufficient funds to cover the specified payments to a receiver?
  3. Is the transaction out of order (i.e., sequence number too low or too high compared to the last received transaction from the same sender)?
  4. Is there sufficient gas or transaction fees to execute the transaction?
  5. Is the transaction oversized?

  If eager validation of a transaction succeeds, the validator pushes the transaction to a pending queue in the transaction pool (this makes a transaction eligible to be included in a block by a validator) and propagates the valid transaction to downstream peers (i.e., validators directly connected to the local node that have not seen the transaction before), eventually propagating the valid transaction throughout the network and having it eagerly validated at every validator. If the eager validation fails, validators drop the invalid transaction.

- **Lazy validation**: Lazy validation occurs before the transactions in a block are executed and checks the sequence number of the transaction, whether the sender account has sufficient balance to send funds to the receiver, and the availability of gas to execute the transaction. Thus, lazy validation is less time-consuming than eager validation.

There is then reasonable doubt whether invalid transactions would be executed, if a malicious validator includes invalid transactions in a block without eagerly validating, and before the execution of the same block at another validator, the lazy validation is not able to capture the invalidity due to the checkers it excludes such as transaction signature verification and transaction size verification. This is a non-issue since the execution of an invalid transaction does not trigger a state transition but will throw an exception. In other words, the state machine after lazily validating transactions rechecks other validity criteria (e.g., transaction signature verification, transaction size limit check) during the transaction execution step, and throws an exception if an invalidity is found. Subsequently, the invalid transaction is discarded [1] [1]. In Chapter 3, we present Smart Redbelly Blockchain [1], that reduces transaction eager validations to improve blockchain performance without impacting security.

2.1.11 Incentives

Public permissionless blockchains [13, 16, 10] provide decentralization by allowing any node to propose blocks to consensus if they expend sufficient resources (i.e., solving PoW puzzle, staking coins). However, without a reward, there is no reason for a node to become a validator and propose blocks expending one's own resources. Thus, public permissionless blockchains reward validators to motivate participation [46] and keep the blockchain decentralized. Rewards can be offered in a number of ways depending on the blockchain implementation. Most often, rewards are offered to validators for proposing blocks that are eventually appended to the final canonical chain. Validators also receive rewards from clients in the form of transaction fees for including client transactions in blocks. The transaction fees are proportional to the gas consumed when executing a transaction. Incentives can also include penalties or punishments in some blockchains [1]. These are negative incentives that validators incur for deviating from the blockchain protocol. Care must be taken when designing a blockchain incentive mechanism to prevent nodes from breaking blockchain safety in order to maximize rewards.

2.2 Blockchain Network models

The network model plays an important role in the guarantees made in a blockchain. We identify below three main network models used in blockchain literature [13, 12, 10].

Synchronous network: Synchronous networks assume that the upper bound on the delay of every message is known [47]. In practice, synchrony is difficult to guarantee and can easily be violated in an open network like the Internet [48].

[1] https://github.com/ethereum/go-ethereum

Partially Synchronous network: Partially synchronous networks assume that there exists an unknown Global Stabilization Time (GST) and a known positive duration δ such that message delays are bounded by δ after GST. In other words, there is no known bound on the transmission delay of messages between nodes [47, 1].

Asynchronous network: Asynchronous networks are the opposite of synchronous networks. More specifically, the upper bound on the delay of every message is unknown. Liveness in blockchains cannot be guaranteed under asynchrony.

2.3 Defining the Blockchain Consensus Problem

2.3.1 Consensus Problem

The Byzantine Generals' problem defines the problem of reaching agreement among a group of processes in the presence of faulty processes [6]. Our consensus problem is based on this notion. A faulty process can either suffer from a crash failure (e.g., hardware failure) or can be Byzantine exhibiting arbitrary behaviour. As mentioned previously, the term *Byzantine* stems from the Byzantine Generals' problem [6] where processes equivocate to cause disagreement. In general, a Byzantine process can exhibit any arbitrary behaviour which can include actions such as equivocation, sending invalid messages or being unresponsive. The opposite of a faulty process is a correct process. Correct processes always adhere to the protocol. In a distributed system, inputting a value to the consensus is known as proposing, and reaching agreement on a value is known as deciding.

A system that requires to reach agreement should satisfy the following three properties [49, 50]:

- **Liveness**: Every correct process eventually decides.
- **Safety**: No two correct processes decide different values.
- **Validity**: If all correct processes propose the same value, no other value is decided.

Solving the consensus problem is required for all distributed systems to reach agreement. Blockchains are also specialized distributed systems requiring consensus to reach agreement. There are however some intricacies in adapting the consensus problem to the blockchain context since blockchains use different primitives such as transactions, blocks, a chain of blocks, and validators. Thus, in the next Section 2.3.2, we reformulate the consensus problem and adapt it to blockchains to define the blockchain consensus problem.

2.3.2 Blockchain Consensus Problem

We reformulate the *blockchain consensus problem* from the consensus problem as the problem of ensuring the liveness, safety, and validity of a blockchain. The safety and liveness properties are adapted from the definition by Garay et al. [51] and the validity property is taken from its classic definition [12].

Definition 1 (The Blockchain Consensus Problem)**.** *The blockchain consensus problem is to ensure that a distributed set of validators maintain a sequence of transaction blocks such that the following properties hold:*

- ***Liveness****: a valid transaction received by a correct validator is eventually reliably stored in the block sequence of all correct validators.*
- ***Safety****: two local chains of blocks maintained by any two correct validators are either identical or one is a prefix of the other.*
- ***Validity****: each block appended to the blockchain of each correct validator contains a set of valid and non-conflicting transactions.*

A valid transaction mentioned in the above properties should adhere to the definition of valid transactions mentioned in Section 2.1.10. The safety property does not require correct validators to share identical chains because one validator may already have received the latest block before another. When the chain is identical at two validators, then the state of these two validators generated deterministically from the blocks in the chain is identical.

Note that when a validator stores a valid transaction within a block sequence, a client asynchronously listening to the response of the particular transaction receives an ACK. Thus the liveness property implicitly provides responsiveness (i.e., the client eventually receives the response from the system).

2.4 Web3

2.4.1 Smart Contracts

Smart contracts are pieces of code often written in high-level languages (e.g., Solidity) that are compiled to bytecode and executed on a blockchain. These pieces of code are uploaded and executed on the blockchain with the use of smart contract upload and smart contract invocation transactions respectively. The execution of smart contracts is duplicated across every node in a blockchain using the instruction set available on each node's virtual machine (e.g., EVM). Smart contracts, as the name suggests, offer conditional payments in the blockchain once certain conditions are met. However, since the inception of the Ethereum blockchain [16], the use of smart contracts has expanded. Rather than executing simple contracts purely for payments, smart contracts nowadays execute as part of Decentralized Applications (DApps).

2.4.2 Decentralized Applications (DApps)

DApps are blockchain applications that consist of a blockchain client that communicates with a smart contract back-end using Remote Procedure Calls (RPC). RPC uses common languages such as JavaScript, Golang, Java, and Rust. A DApp can consist of a group of back-end smart contracts. Non-fungible Tokens (NFTs), Decentralized Finance (DeFi) applications, blockchain games such as crypto kitties, ERC20 tokens, and Decentralized Exchanges (DEX) such as UniSwap are a few common DApps currently in use. The DApp market capitalization was

10.52 Billion USD by 2019[2]. The Ethereum blockchain still remains the largest ecosystem for DApp executions accounting for almost 3000 DApps [3].

2.4.3 Web3

Replacing current web applications with DApps executed on blockchains is touted as the next evolution of the web also known by the buzzword *Web3* [1, 3]. The concept of Web3 removes the inherent dependence on the centralized nature of current web applications. Thus, mitigating a handful of technology companies deciding what should and should not be allowed in the web[4]. The power is given instead to users through decentralization, building a web that is by the people for the people. Unfortunately, as we mention throughout this thesis, there is currently no blockchain capable of supporting demanding realistic DApp workloads to widen the adoption of Web3. As such, we present a new blockchain capable of handling realistic DApp workloads (Chapter 3).

2.5 Blockchain Performance

Blockchain performance evaluations mostly focus on transaction throughput and latency.

2.5.1 Transaction Throughput

Transaction throughput is the number of transactions committed per unit of time by a blockchain. A committing of a transaction refers to a transaction being executed, written to a block, and appended to the blockchain in an irreversible manner. When this happens, blockchains usually generate an ACK (Acknowledgement) notification known as a transaction receipt notifying the client that the sent transaction was committed. Once this ACK notification is received by a client, the client knows that the transaction was successfully committed. When measuring throughput, we consider the client's perspective. In other words, we calculate the number of transactions committed per unit of time from the ACKs a client receives.

2.5.2 Transaction Latency

The transaction latency is the amount of time taken to commit a single transaction. This time is the difference between the transaction send time and the transaction commit time as seen by the client. In other words, the commit time is the time that the client receives an ACK notifying that the sent transaction was committed. Usually, when calculating the latency of blockchains, measuring the latency of a single transaction is not sufficient. Thus, an average of all latencies are taken to measure the latency of a blockchain.

[2] https://www.emergenresearch.com/industry-report/dapps-market
[3] https://www.tap.global/blog/what-are-dapps
[4] https://ethereum.org/en/web3/

2.5.3 Blockchain Congestion

Blockchain congestion occurs when requests are received faster than they can be processed by the blockchain. This leads to the saturation of transaction queues in the blockchain. As a result, transaction losses and performance degradation can be witnessed during periods of blockchain congestion. The performance degradation in such occasions can be identified by a decrease in throughput and an increase in latency.

Blockchain congestion has been a major problem in the blockchain space, largely due to the increase in the number of users, as well as the wider adoption of demanding DApps. Not only is this problem common to the oldest DApp-enabled blockchain, Ethereum [26], but also to one of the most recent and fastest blockchains, Solana [27]. A recent in-depth study demonstrated that, due to congestion, 6 modern blockchains lose transactions and degrade in performance when executing DApps under real application workloads [28].

2.5.4 Blockchain Performance Evaluation Tools

Hyperledger Caliper: Hyperledger Caliper [52] is a blockchain performance evaluation tool. Caliper can evaluate Ethereum and a set of blockchains developed within the Hyperledger project including Besu and Fabric. The blockchain evaluation metrics that Caliper supports include transaction throughput, transaction latency, and resource usage (i.e., CPU, Memory, and Network usage). Caliper supports a pre-defined workload that should specify the calling smart contract, the calling contract function, and the transaction sending rate. The pre-defined workloads in Caliper are synthetic and user-defined and may not provide a realistic evaluation of blockchains in a real-world setting.

Chainhammer: Chainhammer [53] is a blockchain evaluation tool that primarily evaluates the throughput of EVM-based blockchains under demanding workloads. It does not support varying transaction sending rates but evaluates blockchains under a constant high workload. Chainhammer also does not provide flexibility to adjust transaction workloads to mimic realistic scenarios.

Diablo: Diablo [28] is a blockchain benchmarking suite that supports the evaluation of a wide range of blockchains. Currently, Diablo supports the evaluation of Algorand [10], Solana [39], Diem [38], Quorum [11], Ethereum [16], Avalanche [37], and the Smart Redbelly Blockchain [1] presented in this thesis. Diablo implements a generic client interface that can be changed to support the evaluation of many blockchains. Most importantly, Diablo evaluates blockchains under realistic DApp workloads. Diablo mimics transaction workloads observed in the real world for popular web applications like nasdaq, Uber, and fifa to produce realistic workloads for DApps. The key idea is to observe how blockchains perform when evaluated under realistic DApp workloads so that observations can be made on the capability of blockchains to widely support Web3. In this thesis, we use the Diablo blockchain benchmark suite for all our benchmarks to reliably evaluate our contributions in enhancing blockchain performance to widen the adoption of Web3.

2.6 Blockchain Governance

Blockchain governance is the processes relied upon to make decisions and modify the blockchain protocol [2]. More specifically, blockchain governance includes performing key tasks in the blockchain. These tasks may include proposing a block to the blockchain, deciding a block to be executed, changing blockchain parameters such as the number of transactions a block can store, verifying transactions, and upgrading the blockchain protocol to newer versions [54]. Note that a validator is a type of blockchain governor as validators solve consensus (i.e., decide on the order of transactions and blocks to be executed). However, blockchain consensus alone is not sufficient to ensure blockchain governance as it can involve complex decision making processes among stakeholders in a dynamically changing environment.

Blockchain governance is an important concept and the absence of governance in the past has led users to create dissident instances of the two largest blockchains: Bitcoin is now split into BTC and BCH while Ethereum is now split into ETH and ETC [2, 55, 56] (Fig 2.1)

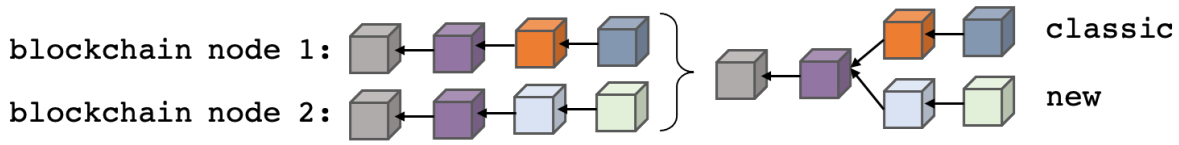

Figure 2.1: If blockchain nodes disagree on a protocol update then they may start accepting distinct blocks, which results in a split with a classic version of the blockchain (e.g., ETC, BTC) and a new version of it (e.g., BCH, ETH).

Since blockchains often consist of thousands of nodes, to achieve scalability, governance tasks are often restricted to a subset of blockchain nodes known as governors [2]. The set of governors are often termed as a committee or a governor committee. There are a number of processes relied upon to elect governors from blockchain nodes to perform key tasks. Below we explain such governance membership selection methods.

2.6.1 Governance Selection

PoW (Proof-of-Work): PoW was first introduced to blockchains in Bitcoin [13] and has since been used in a number of blockchains [16, 57, 58] for various purposes. The main use cases of PoW in blockchains are to (1) select a validator to perform governance tasks such as proposing a block, (2) delay block proposals, and (3) mitigate Sybil attacks where an adversary can assume multiple identities to overwhelm the governor committee [59].

In Bitcoin and many other PoW-based blockchains, a node is required to do some work in the form of solving a computationally expensive cryptographic puzzle to propose a block, a process also known as mining. More specifically, solving the cryptographic puzzle entails finding a number known as a block nonce which when hashed with the contents of a block yields a result that falls within a certain threshold. This threshold is known as the difficulty value. Thus, the only way to find the block nonce that yields the solution is to expensively guess. Once a node finds the block nonce, its verification can be done trivially by checking if the solvers' block nonce

hashed with the block content yields the desired difficulty value.

In PoW-based blockchains, validators compete to solve the PoW for a block. The first validator to do so for a particular block proposes it to the network. Each validator, upon verifying the PoW of the received block, starts working on the next block building upon the previously verified block. The difficulty is adjusted for the next block such that it takes some time for a validator to solve PoW for the next block and propose it to the network. The difficulty of solving PoW is by design. First, as it involves costly guesswork, for any given index in the blockchain, any validator has a probability to propose a block, making the blockchain decentralized. Second, performing PoW slows down block proposals reducing the probability of multiple validators proposing blocks for the same index in the chain creating what is known as a soft fork [60]. In Section 2.8.1, we discuss how such scenarios can be handled by executing blockchain consensus.

PoS (Proof-of-Stake): PoS was developed to address the poor performance and high energy consumption of PoW [61, 62]. Similar to PoW, PoS can select governors to perform specific tasks such as propose blocks. However, in contrast to PoW, PoS does not require a node to solve a complex cryptographic puzzle. Instead, nodes are selected to perform specific tasks proportional to the amount of resources staked by the node. Staking is similar to betting where a deposit is made with the resources a node possesses. If the node misbehaves this deposit is taken, as known as stake slashing in blockchain terms. While theoretically, the stake can be any resource a node possesses, in most cases it is often a monetary asset such as crypto coins or tokens. One limitation of PoS is that it provides more opportunities for those having higher stake to be selected as governors [2]. Given the skewed distribution of wealth, PoS can inadvertently create an oligarchy [63].

DPoS (Delegated Proof-of-Stake): DPoS is a variant of PoS used in EOS, Cardano, and TRON [64, 65, 66]. Instead of staking resources directly to be selected as governors, nodes delegate power to candidate nodes to become governors by voting. Voting nodes stake their assets on candidate nodes. The idea is that an election algorithm (e.g., EOS uses a multi-winner approval voting algorithm [67]) selects the delegates (i.e., governors) where the votes by each voter node to a delegate are weighed by the stake owned by the voter. As PoS-based voting approaches weigh ballots of voters based on the coins they have staked [64, 65, 66], the impact on the election outcome if an adversary splits their stake among multiple identities (Sybil's attack) and cast ballots is minimized [2].

Elected nodes who become governors are selected to propose blocks based on their own stake and the stake backing of voter nodes. When a governor proposes blocks, the reward it gets is distributed proportionally among the voters that elected the said governor. Thus, a voter in their best interest votes for a candidate node that has a higher stake of their own or a significant stake backing such that the chances of the candidate node proposing blocks as a governor are high. If a governor's stake is slashed due to misbehaving, the nodes that backed the governor will not lose a proportion of their stake.

NPoS (Nominated Proof-of-Stake) Another variant of PoS known as NPoS is used in the Polkadot blockchain [54]. It is almost identical to how DPoS works but is different in that (1) voters (i.e., nominators) can lose stake if the governor they elected misbehaves, and (2) upon electing the governors, the stake is evenly distributed among the elected governors such that each governor has an equal chance of proposing blocks. However, both DPoS and NPoS require voters to possess stake to elect a governor. The more stake a group of voters possesses, the more chance they have of electing a set of preferred governors leading to an oligarchy.

Other Governance Selection Methods: There are other methods to select a group of governors in a blockchain. Algorand [10] employs a Verifiable Random Function (VRF) coupled with PoS to select a governor committee, a concept also known as cryptographic sortition. The key advantage of cryptographic sortition is its non-interactive nature that prevents adversaries from predicting future governors. However, given the reliance on PoS, the Algorand governance method can lead to an oligarchy among governors as nodes with more stake have a higher probability of being elected as governors. Proof-of-Authority (PoA) was first introduced as part of the Ethereum blockchain [68] to mitigate the shortcomings of PoW in small networks. Unlike PoW, the PoA implementation of Ethereum also known as clique [5] maintains a list of governors known as an authority and selects one of these governors per *epoch*, which is a period that a static set of governors remain active, to propose a block. The proposed block must be signed with the selected authority's private key which is known as sealing a block. The authority can be either static or dynamic. A static authority is defined in the genesis block when the blockchain bootstraps while the authority can be made dynamic by facilitating existing authority nodes to add or remove authorized members. PoA was recently found to be vulnerable to cloning attacks [69].

2.6.2 Blockchain Governance Reconfiguration

As blockchains typically handle valuable assets, several works already noted the risk of a user bribing other users to build an oligarchy capable of stealing these assets [70]. Having a fixed set of governors can expose these governors to such bribery attacks. Blockchain governance reconfiguration is the process of rotating governance committees periodically to mitigate such bribery attacks. There are several blockchains that use governance reconfiguration to mitigate bribery attacks [10, 30, 31]. Most of these works explicitly assume a slowly-adaptive adversary [71, 10, 31] that can corrupt a limited number of nodes between consensus epochs but cannot corrupt participants during an epoch.

2.6.3 Voting Schemes to Elect Governors

Blockchains that use DPoS or NPoS [64, 66, 65, 54] use voting schemes at their core to elect governors. Both Polkadot [54] and EOS [67] use a multi-winner election protocol to select a group of governors. The former uses a multi-winner election protocol coined the sequential Phragmén method [72]. However, the reliance on PoS favors the wealthiest or users with the

[5] https://eips.ethereum.org/EIPS/eip-225

most resources. Given the Pareto Principle [73] stating that few users typically own most of the resources (as an example, in 2021, the wealthiest 1% of US citizens owned about 1/3 of the total wealth [74]), these approaches have the risk of forming an oligarchy of governors.

In the context of electing governors, a multi-winner election protocol executes as follows: Given a set of n voters, each casting an ordinal ballot as a preference order over all m candidates, a multi-winner election protocol outputs a winning committee of size k.

Proportionality: Election algorithms consist of multiple unique properties. An interesting property in election algorithms is that of proportionality.

Black [75] was the first to define the proportionality problem where elected members in an election must represent "all shades of political opinion" of a society. Dummett [76] later introduced fully proportional representation to account for ordinal ballots, containing multiple preferences. Dummett's fully proportional representation indicates that given a set of voters n voting to elect a committee of k governors, if there exists $0 < l \leq k$, a Hare's quota such that $q_H = n/k$, and a group of $l \cdot q_H$ who all rank the same l candidates in the top of their preference orders, then these l candidates should all be elected [76]. However, the use of the Hare's quota q_H makes Dummett's fully proportional representation vulnerable to strategic voting whereby a majority of voters can elect a minority of seats [77]. This problem was solved with the introduction of Droop's quota q_D as the smallest quota such that no more candidates can be elected than there are seats to fill [78]. Woodall [79] replaces Hare's quota with Droop's quota $q = \frac{n}{k+1}$ and defines the Droop proportionality criterion tweaking the fully proportional representation property: if for some whole numbers j and s satisfying $0 < j \leq s$, more than $j \cdot q_D$ of voters put the same s candidates (not necessarily in the same order) as the top candidates in their preference list, then at least j of those s candidates should be elected.

There are other election methods that do not ensure fully proportional representation such as First-Past-The-Post (FPTP) single-winner election and the Single Non-Transferable Vote (SNTV) multi-winner election [80]. This is because voters can only reveal their highest preference, and only the highest preference is counted in the election outcome.

The fully proportional representation property can be achieved using the Single Transferable Vote (STV) algorithm [78] also used in the Australian Senate. In STV, candidates are added one by one to the winning committee and removed from the ballots if they obtain a quota q of votes.

In chapter 6, we present a governance election method that mitigates an oligarchy of governors by using a *proportional governance* method [2]. More specifically, we adapt the STV (Single Transferable Vote) election algorithm to a Byzantine setting to elect proportionally a diverse set of governors to mitigate an oligarchy being formed in the governance through the election process.

2.7 Blockchain Sharding

2.7.1 Introduction to Sharding

Sharding was a concept first made popular with database systems. Due to the growing amount of requests, sharding became popular to scale databases for fast information retrieval [4]. The sharding technique consists of splitting a database structure across multiple machines called a *shard*. For example, a table can be split into rows or columns and stored across multiple machines reducing the database index at each machine, thus speeding up information retrieval. Similar to database systems, to address blockchain congestion and scalability limitations sharding was applied to a number of blockchains [30, 31, 57, 81, 82]. The basic idea of sharding in blockchain is to split different blockchain tasks into shards. For instance, instead of the entire network of nodes executing consensus, the blockchain network can be split into multiple shards each consisting of validator committees that concurrently solve consensus on disjoint transactions. Similarly, other blockchain tasks such as validating transactions and maintaining blockchain state can be dedicated to multiple shards where each shard performs these tasks on a disjoint set concurrently, helping improve performance. Elastico [71] shards the consensus into committees that process transactions concurrently. OmniLedger and RapidChain [30, 31] shards both the consensus and the blockchain state. Redbelly blockchain shards transaction validations/verifications [12].

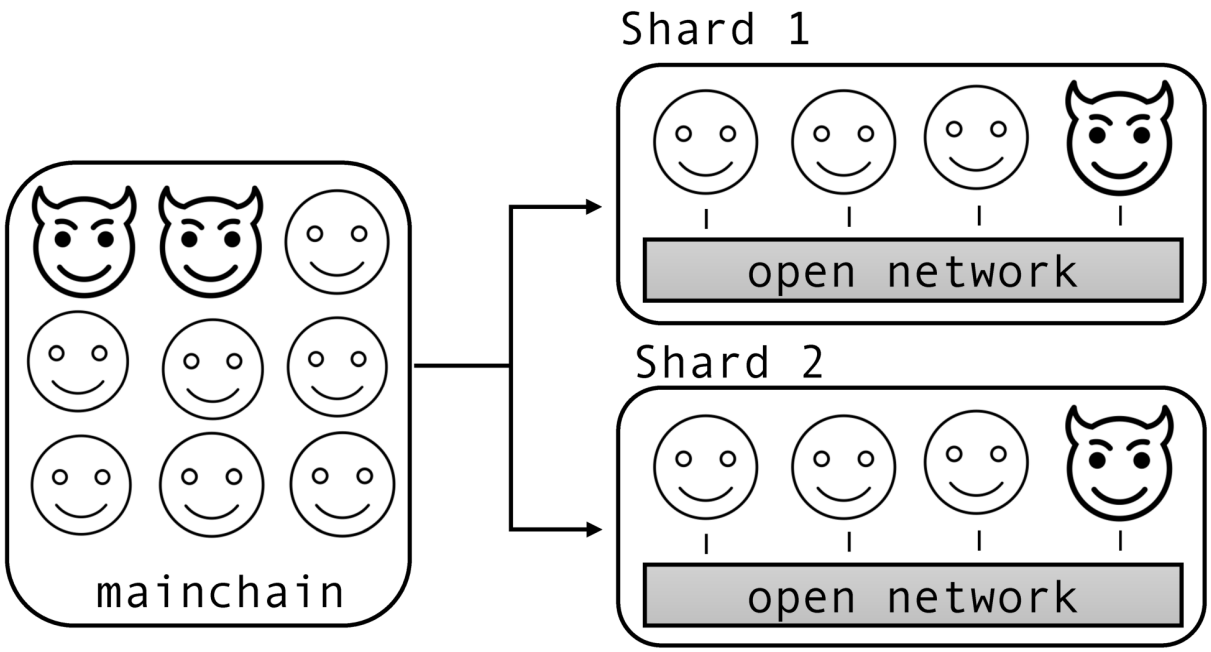

Figure 2.2: Random spawning of shards prevents Byzantine nodes from overwhelmingly joining a single shard.

2.7.2 Deterministic Sharding

Deterministic sharding [83, 84, 12] consists of assigning blockchain nodes or transactions to shards deterministically. The advantage of such an approach is that the current shard state

is inherently transparent as anyone can infer the shard of a node or transaction by simply computing a local deterministic function.

SharPer [83] creates shards deterministically based on geographical distribution. Nodes that are located close to each other are assigned to the same shard. Red Belly Blockchain [12] shards only the transaction verification (i.e., validating that a transaction is signed properly). The motivation stems from the fact that the verification of cryptographic signatures is CPU intensive. Instead of having all nodes verify all transactions, Redbelly Blockchain assigns deterministically each transaction, based on its hash, to two subsets of nodes: its $f+1$ primary verifiers and its f secondary verifiers [12]. The secondary verifiers wait for some time for the primary verifiers to verify the transaction. If the primary verifiers are too slow or unresponsive, then the secondary verifiers start verifying the transaction. A node detects whether a transaction is properly signed once it receives the same response from $f+1$ distinct verifiers.

The drawback of deterministic sharding is that the outcome of the deterministic sharding function is predictable, which makes the blockchain vulnerable to attacks. In particular, an attacker can exploit this information to overwhelmingly join a single shard and prevent the validators of the said shard from reaching consensus, potentially leading the blockchain to an inconsistent state. Such inconsistent states cause "forks" that are exploited by various attacks [85, 86] to double spend. SSChain [84] allows nodes to freely join a shard deterministically. To mitigate shard-takeovers [3] SSChain follows a two-chained approach: a root chain verifies the blocks coming from each shard before committing them, and a shardchain agrees upon blocks to send to the root chain. The root chain is able to make an accurate verification of shard blocks by storing the full state of the blockchain.

2.7.3 Probabilistic Sharding

Most blockchain sharding approaches use a set of governors known as a mainchain or a main committee (i.e., also known as the beacon chain in Ethereum [82]) to spawn shards [3, 57]. Previously mentioned governance selection methods such as PoW and PoS are used to first select governors to the main committee and to mitigate Sybil attacks. Subsequently, the main committee creates shards randomly from blockchain nodes. The use of randomness to create shards prevents a Byzantine node from foreseeing the shard they will join. This mitigates Byzantine nodes overwhelmingly joining a single shard to corrupt it (Fig 2.2) and cause double spending (Section 2.9.3). Once a shard is created, the nodes in each shard act as a committee of validators independently solving consensus, and deciding on disjoint transactions. However, since the number of validators in each shard is few in number compared to a non-sharded counterpart, the risk of bribery attacks (Section 2.9.2) rises. This stems from the logic that it is often easier to bribe a few validators than a considerable number. If sufficient validators in a shard are bribed, consensus disagreements occur leading to double spending attacks (Section 2.9.3). To mitigate this, most sharding approaches reconfigure committees periodically similar to governance reconfigurations mentioned previously [54, 10, 71, 57].

2.7.4 Limitations

Cross-shard transactions are specific types of transactions that update state in two or more shards. As one shard may have to wait for a withdrawal or deposit part of a transaction to be completely executed in another shard before executing the transaction itself, the performance can be impacted. With an increasing number of cross-shard transactions, the performance gains of sharding can be limited [3, 4].

Most sharding approaches are static [3] where the number of shards and the number of nodes within a shard cannot be adjusted at runtime [30, 31]. As blockchains are built to execute for a long period (e.g., Bitcoin [13] has been running for more than a decade without interruption), and since the transaction request rates to blockchains change over time, there is a need to dynamically adjust shards according to demand. Ideally, during periods of high demand, the blockchain should execute more shards to reduce blockchain congestion, whereas during low demand the blockchain should reduce the number of shards to reduce operating costs (e.g., electricity charges).

Another limitation with most sharding protocols is their inability to provide clients with the sharding configuration transparently and securely [3].

In Chapter 4, to alleviate the aforementioned limitation, we present a dynamic transparent DApp-oriented blockchain sharding approach.

2.8 Blockchain Consensus Protocols

For the first public permissionless blockchains like Bitcoin and Ethereum [13, 16], a consensus protocol known as Nakamoto's consensus was used [13]. Later, BFT consensus algorithms were adopted for permissioned blockchains with some changes in the blockchain architecture [46]. These BFT algorithms are now being used in public permissionless blockchains as well [87, 10, 38, 37].

2.8.1 Nakamoto's Consensus

Nakamoto's consensus was first introduced in the Bitcoin blockchain [13] and is a namesake for Satoshi Nakamoto, the author of the Bitcoin white paper [13]. Nakamoto bought forth the idea of reaching consensus on the longest chain in the event that multiple chains of blocks have formed due to network delays between validators. The longest chain rule defines that the chain of blocks with the highest indexed block should be selected as the final canonical chain [46].

In the context of Bitcoin, the longest chain rule helps achieve consensus. In Bitcoin, a validator broadcasts a block after performing PoW. Each validator receiving the proposed block performs a verification which includes verifying the PoW for the block. If the verification is successful, validators append the received block to its relevant parent block. Due to network delays, there can be instances where two or more validators propose blocks for the same index in the chain without knowing another validator has solved PoW for the same block index. In such instances, validators may receive two blocks that are both valid for the same index of the chain. When this happens validators append both blocks to the same index creating what is

known as a fork. This fork is resolved through the longest chain rule eventually deciding a single final chain. Once the fork is resolved, the blocks and the transactions at a particular height from the tail of the chain (e.g., 5 blocks from the tail) are considered confirmed. This confirmation is done with a high probability that another longer chain would not exceed the current longest chain such that the block at a particular height is overwritten. While under synchrony a longer chain will not be revealed after an unknown time guaranteeing liveness, under a realistic network setting such as the internet, where the communication is asynchronous or at best partially synchronous, a longer chain can be revealed after sufficient time, overwriting the current longest chain. Thus, you can never guarantee that consensus has been reached at a given index of the blockchain. As such, probabilistic consensus does not guarantee either safety under asynchrony or partial-synchrony.

Another downside to the longest chain rule is the possibility of forks being formed. Not only will this impact performance due to having to resolve forks constantly, but it may also lead to blockchain attacks and potentially double spending [88, 89] (Section 2.9.3). The Bitcoin blockchain minimizes the formation of forks with the use of PoW. By requiring validators to perform work that takes a time larger than the block propagation time, Bitcoin rate limits the number of blocks proposed per index.

2.8.2 BFT Consensus

To mitigate the shortcomings of Nakamoto's consensus, such as high resource usage, low performance, and large carbon footprint, BFT consensus algorithms were introduced to permissioned blockchains [11]. BFT consensus algorithms can satisfy the consensus properties of liveness, safety, and validity when in the presence of Byzantine faults. Furthermore, BFT consensus algorithms do not create forks and deterministically reach finality on a value (i.e., a block in the context of blockchains) [36]. However, popular BFT consensus algorithms are known to scale poorly for a large number of processes without architectural changes [90, 46]. BFT consensus algorithms started becoming feasible in blockchains with the advent of committee-based blockchain protocols. These committee-based protocols periodically selected a subset of nodes known as a committee to solve consensus [10, 54, 37]. A relatively small number of nodes solving consensus allowed BFT algorithms to be used without impacting blockchain scalability. Risks introduced in committee-based blockchain protocols such as the vulnerability of committees against bribery attacks (Section 2.9.2) were mitigated by reconfiguring/rotating committees (i.e., governance committee) periodically [10, 54]. Sybil's attacks were mitigated by using stake-based weights on votes when selecting a committee. Once BFT consensus algorithms became feasible in blockchains, a number of blockchains started employing such algorithms to minimize resource usage, carbon footprint and improve performance [10, 54, 37, 11]. Below we discuss several BFT consensus algorithms used in a number of popular blockchains.

Practical Byzantine Fault-Tolerant (PBFT): PBFT is a leader-based BFT consensus algorithm that tolerates f faulty nodes such that $3f < n$ [87]. PBFT elects a leader known as a primary from a set of processes for a specific period of time known as a view. Upon receipt

of a client's request via a point-to-point message, the primary initiates a three-phased protocol to ensure that all processes agree on the order of transactions within a view and across views, and satisfies the safety property of the consensus problem (Section 2.3.1). The three phases of PBFT are *pre-prepare*, *prepare*, and *commit*. First, a primary multicasts a *pre-prepare* message to all processes. Upon receiving a *pre-prepare*, a process verifies the message, and if successfully verified sends an acknowledgment in the form of a *prepare* message containing the contents of the *pre-prepare* message. Upon a process receiving $2f$ *prepare* messages from unique processes and a corresponding pre-prepare message, the *commit* phase commences. In this phase, processes send a *commit* message to other processes. Finally, processes commit transactions, upon receiving $2f + 1$ *commit* messages with the same value each from a unique process. Leader-based BFT consensus algorithms like PBFT have a view-change protocol to elect a new primary (leader) if the current primary is unresponsive or faulty to ensure the progress of the system. The communication complexity of PBFT in the normal case is $\mathcal{O}(n^2)$.

A few noteworthy blockchains such as Solida [91], Hyperledger Fabric [33], and ByzCoin [92] uses variants of PBFT consensus.

Istanbul BFT (IBFT): IBFT is a BFT consensus algorithm that features the same three-phased commit protocol of PBFT consisting of pre-prepare, prepare, and commit [93]. In the good case, IBFT terminates in three message delays and has a communication complexity of $\mathcal{O}(n^2)$. As with PBFT, IBFT tolerates up to f faulty nodes such that $3f < n$. However, PBFT and IBFT have two clear distinctions. First, unlike PBFT, IBFT does not restrict servers (i.e., validators in blockchains) from proposing blocks. In fact, both validators and clients can propose blocks. Second, IBFT supports a dynamic set of validators whereas PBFT requires a static set. IBFT is currently used in the Quorum blockchain, which is a permissioned blockchain featuring the EVM. A recent analysis of IBFT identified that it does not satisfy liveness in a partially synchronous network, leading to a more improved version known as IBFT2.0 which is said to satisfy both safety and liveness [94].

BA⋆ (Byzantine Agreement): BA⋆ is a BFT consensus algorithm used in Algorand [10] that achieves scalability by selecting a committee of nodes to solve consensus based on a VRF. With BA⋆, the number of nodes executing consensus is dynamic and a reconfiguration of committees occurs periodically making it suitable for a permissionless setting. To prevent Sybil attacks, BA⋆ weighs users according to their stake in the consensus committee selection process where higher staked users have a higher probability of being selected to the committee. This makes it difficult for an adversary to split their stake among multiple identities and gain a majority coalition in the consensus committee. As with classic BFT consensus algorithms [6], BA⋆ also requires the number of faulty nodes f to be less than a third of the total nodes (i.e., $3f < n$). BA⋆ consists of two phases. The first phase includes a leader proposing a block that is reduced to a binary consensus (i.e., each process inputs a 1 or 0 value to the consensus and decides on either 1 or 0). The second phase makes the binary decision on the proposed block based on voting. BA⋆ requires a strong-synchrony assumption to achieve safety. Finally, BA⋆ terminates in 6 rounds in the normal case and 13 rounds in the worst case [10, 46].

DiemBFT: DiemBFT is a leader-based BFT consensus used in the Diem blockchain [38] that operates in a permissioned network. As such, one must receive approval from Diem Networks[6] to be validator nodes in the committee. The core principles of DiemBFT rely on the classical BFT approach [6]. As such, DiemBFT tolerates up to f faulty nodes such that $3f < n$. The commit protocol of DiemBFT is inspired by HotStuff [95] and follows the phases of *prepare*, *pre-commit*, and *commit*, in which each phase creates a *quorum certificate* consisting of votes from $n - f$ nodes for a block and their respective signatures. The main novelty of DiemBFT is its leader election mechanism. Leader election in DiemBFT is done symmetrically in order to ensure fairness. To prevent crashed leaders from being elected DiemBFT introduces a leader utilization mechanism that leverages a reputation scheme to limit the number of times a crashed leader can be elected. In case a faulty leader is elected, DiemBFT triggers a quadratic view change protocol. In a network with no faulty leaders, DiemBFT achieves linear communication cost similar to HotStuff [96]. In the worst case where faulty leaders exist, DiemBFT achieves a cubic communication cost similar to PBFT [38].

Avalanche consensus: Avalanche [37][7] introduces a new BFT consensus algorithm that can adjust the number of Byzantine failures the network can tolerate by tweaking certain parameters. Its core idea relies on a protocol known as *Snow*. On a high level, a node participating in the Snow protocol selects a sample of nodes from the network and queries them for a transaction. If a quorum within the selected sample sends the same transaction, the receiving node adapts the transaction value, and increments the success counter. In the next round, when nodes query transactions again, their adopted transaction in the previous round is sent by each node. After a threshold of consecutive successes on a particular transaction across multiple rounds, each node decides the said transaction. As the Snow protocol allows nodes to converge to a single decision, if one correct node decides a certain transaction, the same transaction will be decided by all other correct nodes. The use of constant samples makes the Snow protocol scale to large networks as the number of messages sent remains constant despite the growth in the total number of nodes. Unlike classic blockchains [13, 16, 10, 38, 54, 11, 43], Avalanche stores transactions in Directed Acyclic Graphs (DAGs), with a confidence value analogous to the concept of confirmations for a block appended to the blockchain in Ethereum and Bitcoin.

Democratic Byzantine Fault Tolerant (DBFT): DBFT [36] is a leaderless BFT consensus algorithm that assumes partial synchrony. All nodes executing DBFT can propose blocks for a consensus round making the decision inherently "Democratic". Unlike conventional leader-based consensus algorithms that assume partial synchrony, DBFT consists of a weak coordinator that proposes its value without imposition. As all nodes can propose a block, nodes can decide a value even if the coordinator becomes slow or faulty. In the best case, when all non-faulty processes propose the same value, DBFT can terminate in 4 message delays. The DBFT consensus is used in the Redbelly blockchain [12]. To improve performance, the DBFT consensus in the Redbelly blockchain uses an optimization known as the superblock. Moreover, the DBFT

[6] https://www.diem.com/en-us/

[7] https://docs.avax.network/overview/getting-started/avalanche-consensus

consensus decides a vector of values per consensus round rather than a single value. This vector of decided values (i.e., vector of blocks) is then merged removing any conflicting and invalid transactions in a process known as reconciliation [36]. The resulting batch of transactions output from reconciliation is known as the superblock and is sent to the SM for execution. The Smart Redbelly Blockchain we present in this thesis uses the Redbelly blockchain's consensus with some distinctions. Unlike Redbelly blockchain, SRBB introduces an important transaction validation reduction and supports DApp executions. We outline these details in Chapter 3.

Scalable Byzantine Fault Tolerant (SBFT): SBFT [97] is a byzantine fault-tolerant consensus algorithm that builds upon PBFT but achieves better scalability by using threshold signatures to reduce the communication complexity [5]. Like PBFT, SBFT commits at most one proposed block per consensus instance. The Concord [98] blockchain combines a lightweight C++ implementation of the EVM with SBFT.

2.9 Blockchain Attacks

While blockchains can be vulnerable to a multitude of attacks depending on their implementation, in this thesis we turn our focus to three common attacks when presenting our contributions.

2.9.1 Sybil Attacks

A Sybil attack [99] consists of an adversary impersonating multiple identities to overwhelm the blockchain and cause disagreements in governance including consensus. For example, if an adversary impersonates sufficiently many validators, they can form a coalition in the consensus committee causing either disagreement (i.e., forks), which impacts blockchain safety, or remain passive impacting blockchain liveness. As mentioned previously, the use of PoS and PoW approaches mitigates Sybil's attacks on blockchains as they make it difficult for an adversary to assume the identity of multiple users. In PoS approaches, an adversary has to split their stake to impersonate multiple users but splitting stake can reduce the chances of being selected to perform a specific governance task. This is because in PoS approaches the probability of being selected to perform tasks is proportional to the stake owned. In PoW approaches, the adversary has to split their computing power to assume the identity of multiple users. This effectively reduces the hash power behind each impersonated user, reducing the probability of such users being selected to the committee.

2.9.2 Bribery Attacks

A bribery attack is the act of offering something to corrupt a blockchain node (e.g., a validator). If a sufficiently large number of validators are bribed (e.g., f s.t. $f \geq n/3$ in blockchains with BFT consensus), a blockchain validator/governance committee will consist of a coalition of corrupt validators causing forks in the system. This can impact both the safety and liveness of the blockchain.

2.9.3 Double Spending Attacks

Double spending attacks, as its name suggests, occur when the same coin or token is spent twice in a blockchain, allowing a user to purchase more assets for the price of a coin or token. An adversary launches a double spending attack by causing a fork in the blockchain consensus using a threshold coalition. In blockchains with BFT consensus, if the number of validators is n, the threshold coalition is f s.t. $f \geq n/3$. To gain control of a threshold coalition, an adversary often launches bribery or Sybil attacks on the blockchain.

Chapter 3

Smart Redbelly Blockchain: Reducing Congestion to Improve Performance

In this chapter, we present Smart Redbelly Blockchain (SRBB), a permissionless blockchain that enhances blockchain performance for DApp executions with the motivation of widening the adoption of Web3.

Decentralization promises to remedy some of the weaknesses of the web, including data exposure [100], user manipulation [101], and outages [102]. The idea, often called Web3, is to execute Decentralized Applications (DApps) on blockchains. Unfortunately, Web3 remains hypothetical as blockchains suffer from performance degradation and transaction losses induced by congestion from realistic DApp workloads [26, 27, 28]. In fact, Ethereum, the largest DApp-compatible blockchain in market capitalization, has experienced congestion for at least 5 years since Initial Coin Offerings (ICOs) [103] and CryptoKitties [104] became popular. These congestion issues are not just isolated to Ethereum since they recently affected a supposedly fast blockchain called Solana [27]. Recent benchmarks also indicate that 6 modern blockchains lose transactions when executing realistic DApp workloads [28] with high transaction request rates. Therefore, to decentralize the web, one has to first address the problem of blockchain congestion.

We identify two important causes of congestion in modern blockchain designs [10, 37, 38, 16, 11, 39] that leads to transaction losses and performance degradation. First, every transaction is propagated across the network of validators, and validated at each validator leading to as many validations as there are validators making the transaction validation redundant. After the validation of transactions, validators in many modern blockchains [38, 16, 11] include transactions in blocks and propagate them again as part of a block across the network. Thus, transactions are propagated redundantly, first individually and then in blocks. Modern blockchains propagate transactions individually to validators prior to proposing them in blocks to increase the probability of a transaction being included in a block immediately, as not every validator gets to propose a block per consensus round [16, 10, 11]. Removing the initial propagation of transactions between validators would require the first validator receiving client transactions to

include it in a block. Until the validator that receive a client transaction is selected to propose a block through consensus, a client can expect to wait a considerable time. Thus, the initial propagation of transactions is necessary for modern blockchains to prevent significant processing times of client transactions. The redundant transaction validation is a consequence of the initial transaction propagation. As invalid transactions should not be propagated, every correct validator validates a transaction prior to propagation. As we see in Section 3.5, this initial redundant validation and propagation of transactions causes congestion leading to transaction losses and performance degradation. Second, malicious (or *Byzantine*) validators can include invalid transactions in blocks and broadcast such blocks to the network of validators. This can spam the network with invalid transactions in what we call a *flooding attack* which can cause unnecessary consumption of node resources and network bandwidth, leading to congestion and transaction losses (Section 3.5.3).

We present Smart Redbelly Blockchain (SRBB), a permissionless blockchain that enhances blockchain performance by reducing the two aforementioned causes of congestion. To reduce congestion and enhance performance, SRBB introduces two novel contributions: (1) Transaction Validation and Propagation Reduction (TVPR) and (2) Reward-Penalty Mechanism (RPM). TVPR does not propagate transactions individually among nodes but only propagates transactions in blocks (Section 3.2), hence preventing the redundant validation and propagation of transactions without impacting the core blockchain properties of safety, liveness, and validity (Definition 1). We explain in Section 3.6 that applying TVPR can be problematic to modern blockchains as it can lead to considerable wait times for a transaction to be included in a decided block and executed as not every validator's block is committed per consensus round. However, applying TVPR to SRBB does not cause the same issue due to SRBB's consensus protocol that combines blocks proposed by all validators per consensus round to a decided superblock that is executed [12]. We acknowledge that TVPR could cause censorship of transactions if the validator initially receiving the transaction is Byzantine, and propose methods to alleviate such an issue in Section 3.6. To prevent congestion induced by Byzantine validators performing flooding attacks with invalid transactions, SRBB introduces RPM.

While the name Smart Redbelly Blockchain is derived from its predecessor the Redbelly Blockchain (RBBC) [12], the two blockchains are different. More specifically, Redbelly is an open blockchain, works on an Unspent Transaction Output (UTXO) model, and does not support DApp execution. In contrast, SRBB is a permissionless blockchain, works on an account balance model, supports DApp execution, and features two novel contributions to reduce blockchain congestion and enhance performance: (1) Transaction Validation and Propagation Reduction (TVPR) and (2) Reward-Penalty Mechanism (RPM). These contributions enable SRBB to better support realistic DApp workloads compared to 6 modern blockchains. Namely, Algorand, Avalanche, Diem, Ethereum, Quorum, and Solana [10, 37, 38, 16, 11, 39].

In summary, this chapter presents the following contributions:

- A permissionless blockchain, Smart Redbelly Blockchain (SRBB), that (i) prevents the redundant validation and propagation of transactions found in modern blockchains and (ii) mitigates the propagation of invalid transactions by malicious validators. SRBB uses

TVPR for (i) and RPM for (ii). We prove that despite introducing TVPR and RPM, SRBB solves the blockchain consensus problem, ensuring liveness, safety, and validity (Theorem 1).

- To better assess the advantage of TVPR and RPM, we compared SRBB with a baseline, which is a "naive" smart contract supported version of RBBC consisting of the DBFT consensus [36] with the superblock optimization [12] applied to the Ethereum Virtual Machine (EVM) but without TVPR and RPM. Our results show that TVPR increases the throughput of the baseline by 55× and divides the baseline latency by 3.5 (Section 3.5.2). RPM increases the throughput of the baseline by 7% under flooding attacks (Table 3.1). Lastly, both TVPR and RPM reduce transaction losses.

- To demonstrate the improvements of SRBB over modern blockchains in a common and fair setting, we used the DIABLO benchmark suite [28] with its recommended settings and real DApp workloads. We compared SRBB's performance against the reported performances of Algorand, Avalanche, Diem, Ethereum, Quorum, and Solana. When deployed across the globe to execute a demanding FIFA web service workload on a DApp, SRBB commits twice as many transactions compared to the evaluated 6 modern blockchains in DIABLO. More interestingly, SRBB is the only blockchain out of the evaluated blockchains in DIABLO to commit all transactions for the application workloads of NASDAQ and Uber.

Chapter Outline: The remainder of this chapter is structured as follows: Section 3.1 presents the problems in modern blockchains. Section 3.2 presents SRBB (with TVPR and RPM) and proves it correct. Section 3.5 evaluates SRBB and Section 3.6 discusses it. Finally, Section 3.7 concludes the chapter.

3.1 Problems in Modern Blockchains

In this section, we define the redundant eager validation and propagation of transactions problem, and the invalid propagation of transactions problem in modern blockchains. These problems cause congestion and thereby transaction losses and performance degradation.

3.1.1 Redundant Eager Validation and Transaction Propagation

Redundant eager validations: Many modern blockchains (e.g., Ethereum, Solana, Algorand, Avalanche, Quorum, and Diem) [16, 105, 11, 39, 54, 37] follow a protocol where validators first eagerly validate transactions received from clients or peer validators, add these transactions to their transaction pool pending queue if valid (Alg. 1, line 8) and propagate each valid transaction individually to the network of validators. In other words, every transaction in the blockchain initially gets eagerly validated at every validator node and propagated individually throughout the blockchain network of validators. More precisely, if the number of validators is n, a transaction is eagerly validated n times instead of once. The eager validation of transactions at every validator except the first validator receiving the transaction is redundant, as

transactions are lazily validated (Chapter 2, Section 2.1.10) prior to execution. Even if lazy validation succeeds for an invalid transaction due to lazy validation being weaker than eager validation, the transaction will fail at execution time by throwing an error (Alg. 1, line 32). Thus, the invalid transaction will have no impact on the blockchain state. In fact, we prove SRBB preserves liveness, safety, and validity without the redundant eager validation of transactions (Section 3.4). We discuss in Section 3.6 that eagerly validating a transaction once is sufficient to mitigate DoS attacks.

Redundant transaction propagation: In some modern blockchains [16, 105, 11, 39, 54], transactions from the transaction pool pending queue of validators get included in blocks and propagated again as a part of a block after the initial transaction propagation. In this case, the initial propagation of individual transactions among validators is redundant since transactions are propagated again in blocks. However, the initial propagation of individual transactions ensures a transaction is included in a block without a significant wait time. This is because in modern blockchains only a single validator gets to propose a block per consensus round which is eventually committed and executed [16, 105, 11, 39, 54], and without the initial transaction propagation, a client's transaction may not be included in a block for some time if the validator directly receiving the transaction from the client is not selected to propose a block for an immediate consensus round. In contrast, even when the initial transaction propagation is removed on SRBB, a client's transaction can be immediately included in a block without a wait time as SRBB's consensus allows all validators to propose a block per consensus round, which is combined into a superblock and executed.

In our evaluation (Section 3.5), we show the congestion impacts of the redundant eager validation and propagation of transactions. We observe transaction losses and performance degradation (i.e., throughput and latency) in modern blockchains under realistic DApp workloads.

3.1.2 Invalid Transaction Propagation

The invalid transaction propagation problem occurs when Byzantine validators propagate invalid transactions in blocks. This can happen when Byzantine validators falsely eagerly validate or skip the eager validation of transactions received from clients and peers but include such invalid transactions in blocks nonetheless and propagate them to the network of validators. Propagating invalid transactions in blocks does not cost anything to the validator but can cause transaction losses and performance degradation in modern blockchains due to the following reasons: (1) validators use extra CPU cycles validating invalid transactions (prior to execution) without contributing to the throughput and (2) the network bandwidth is consumed unnecessarily.

While mechanisms to ban validators that propagate invalid transactions can be implemented, such methods can be ineffective. For example, it is unclear whether banning validators is effective. Since banning validators include blocking a validator's address, a banned validator can easily derive a new wallet address and participate in the blockchain again [106].

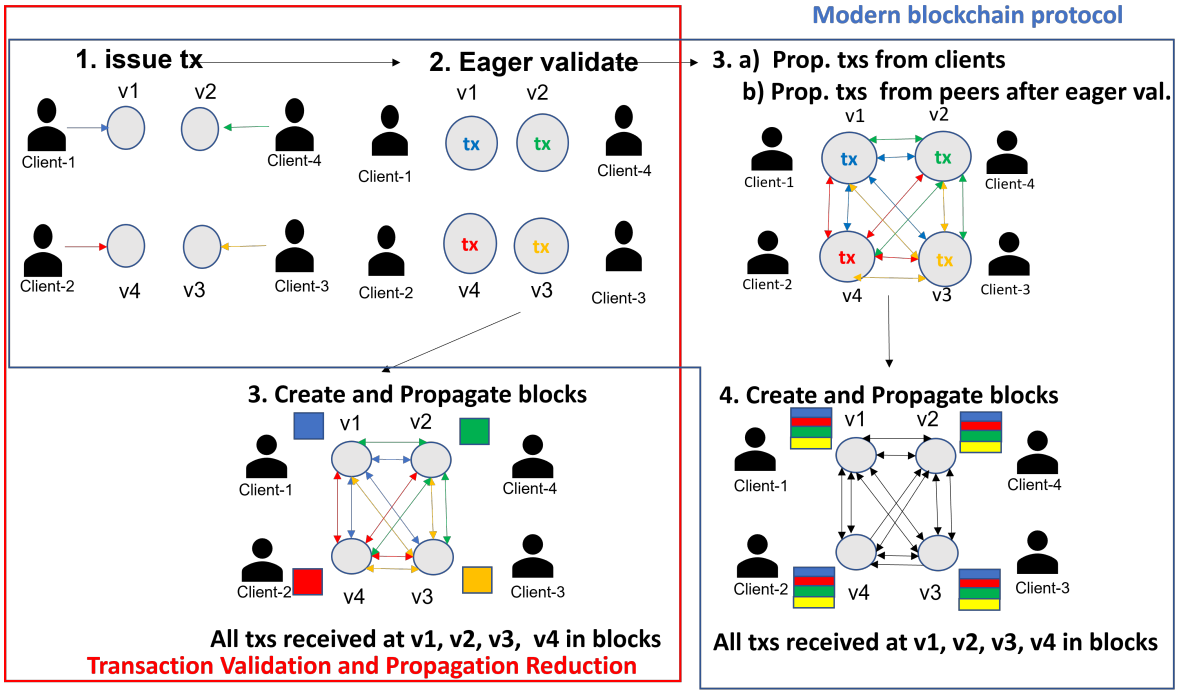

Figure 3.1: The modern blockchain protocol (steps 1-4) and TVPR modification of SRBB (steps 1-3) are represented graphically on a high level. Despite having TVPR, each validator of SRBB executes all transactions due to its superblock-enabled DBFT consensus that combines blocks from each validator to a superblock prior to execution (explained in Section 3.2.4).

Therefore, modern blockchains do not offer validator banning mechanisms in their implementation [16, 11, 44, 43, 39]. There are, however, frameworks like *flashbots*[1] that can integrate validator bans in blockchains. Flashbots is known to be centralized [107]. This was also revealed by its latest Tornado Cash censorship[2].

In Section 3.5.3, we notice transaction losses and performance degradation in modern blockchains caused by congestion when Byzantine validators propagate invalid transactions in blocks. Thus, to mitigate Byzantine validators proposing invalid transactions, in Section 3.2.5, we present a novel RPM that slashes validators for propagating invalid transactions in blocks and rewards validators for proposing valid blocks. Unlike conventional validator banning methods like flashbots[3], RPM is decentralized and does not suffer from censorship issues. This is because RPM punishes validators for including invalid transactions in blocks using a decentralized smart contract-based method. While in conventional validator banning methods, validators can derive new wallet addresses and rejoin the blockchain, the slashing of stake integrated into RPM disincentivizes validators to rejoin the blockchain with new wallet addresses and re-propagate invalid transactions.

[1] https://github.com/flashbots/block-validation-geth
[2] https://www.coindesk.com/tech/2022/08/17/tornado-cash-fallout-can-ethereum-be-censored/
[3] https://time.com/6223034/ethereum-merge-sanctions-flashbots/

Algorithm 1 Smart Redbelly Blockchain protocol

State:
  $blockchain$, an array of blocks, initially:
    $blockchain[0] =$ genesis-block
  $\mathbb{P}$ is the txpool pending queue

receive(t): ▷ *t received from neighbors or directly from clients*
  if eager-validate(t) **then** ▷ *if eager validation succeeds*
    if $t \notin blockchain$ and $t \notin \mathbb{P}$ **then** ▷ *t not in blockchain or txpool pending queue*
      $\mathbb{P} \leftarrow \mathbb{P} \cup t$ ▷ *add to txpool pending queue*
      if TTL of t not exceeded **then**
        propagate(t) ▷ *modern blockchains do this but SRBB does not*

propose():
  $b_i \leftarrow$ create-block-with($\mathbb{Q}$) ▷ *create a block b_i from $\mathbb{Q} \subset \mathbb{P}$*
  $blockQueue \leftarrow blockQueue$.append($b_i$) ▷ *append to block queue*
  $\mathbb{P} \leftarrow \mathbb{P} \setminus \mathbb{Q}$ ▷ *remove txs from txpool to create b*
  $b_i \leftarrow blockQueue[0]$ ▷ *get first element from block queue*
  propagate(b_i) ▷ *propagate block via rb-broadcast*

Upon reception of $\mathbb{B}$ **for index** k **s.t.** $\mathbb{B} \leftarrow \bigcup_{i=1}^{j} b_i$: ▷ *rb-delivered blocks*
  $\mathbb{B}^* \leftarrow$ consensus($\mathbb{B}$) ▷ *exec. DBFT cons. and decide $\mathbb{B}^*$ s.t. $\mathbb{B}^* \subseteq \mathbb{B}$*
  for all $b_i \in \mathbb{B}^*$ starting from $i = 1$ **do**
    for $t \in b_i$ **do**
      $err \leftarrow$ execute(t)
      if $err \neq$ null **then**
        $b_i \leftarrow b_i - t$ ▷ *remove invalid t from b_i*
    if $b_i \neq$ null **then** ▷ *b_i has transactions*
      $blockchain[k] = b_i$ ▷ *insert to permanent chain*
      k++
  $blockQueue$.remove($\mathbb{B}*$) ▷ *remove the set of decided blocks from the block queue if they exist in the queue*

execute(t):
  $err \leftarrow$ lazy-validate(t) ▷ *lazy validation*
  if $err \neq$ null **then**
    return err
  $S_r, err \leftarrow$ ApplyTransaction(t, S_i) ▷ *Apply t on state S_i*
  if $err \neq$ null **then**
    return err
  else
    return null

3.2 Smart Redbelly Blockchain (SRBB)

In this section, we present SRBB, a permissionless blockchain that (1) prevents the redundant validation and propagation of transactions and (2) mitigates the propagation of invalid transactions. SRBB integrates (i) a novel TVPR to prevent the redundant validation and propagation of transactions, and (ii) a novel reward-penalty protocol to mitigate the propagation of invalid transactions. In addition, SRBB is compatible with the largest ecosystem of DApps, optimally resilient against Byzantine failures, and supports real DApp workloads like NASDAQ, Uber, and FIFA (Section 3.5).

First, we present our assumptions followed by TVPR which is a part of SRBB that prevents the redundant eager validation and propagation of transactions problem in modern blockchains. Second, we present the transaction life cycle of SRBB followed by the SRBB Virtual Machine (VM) implementation. Third, we present the novel reward mechanism of SRBB coined RPM that mitigates the invalid transactions propagation problem in modern blockchains. Finally, we prove SRBB solves the Blockchain Consensus Problem (Chapter 2, Sec. 2.3.2).

3.2.1 Assumptions

Partially Synchronous Model: Our network consists of a set of validators n that are well-connected. As consensus cannot be solved in the asynchronous setting, we assume *partially synchronous* communication similar to prior work [47] (Chapter 2, Section 2.2). In practice, we cope with partially synchronous communication by increasing timeouts [12].

Byzantine Model: We assume that out of n SRBB validators, at most f are *Byzantine* where $f < n/3$ (consensus is unsolvable in this model if $f \geq n/3$). The maximum number of Byzantine validators is thus $t = n/3 - 1$ such that $f \leq t$. Byzantine validators can act arbitrarily like proposing conflicting[4] blocks, and invalid transactions. In general, any behavior that deviates from the protocol is considered Byzantine behavior.

Threat Model: Several blockchains [31, 30, 91] reconfigure their committee of n validators every epoch (i.e., a pre-specified unit of time) to mitigate a majority of the committee from being bribed by a slowly-adaptive adversary. A slowly-adaptive adversary is defined as a malicious entity that can bribe validators progressively (not instantaneously) and only between epochs (not during an epoch) such that the total corrupted validators is f where $f < n/3$ at any time [30]. As n validators cannot reach consensus if $f \geq n/3$ validators are corrupt through bribery, the assumption of a slowly-adaptive adversary prevents consensus disagreements and double spending attacks. Therefore, we assume that the adversary is slowly adaptive similar to prior works [31, 30].

[4]Two transactions can conflict if they read the same data and one transaction is a write request [41]

3.2.2 Membership and Committee Reconfiguration

SRBB is a permissionless blockchain where (1) any node can join or leave the network and (2) any node has a chance to become a validator based on a periodic election process if they stake a certain amount of tokens. This membership approach of periodically rotating validators prevents an exclusive set of nodes from always being validators, providing a permissionless environment similar to Algorand [10]. An SRBB node can be (1) a client that sends transactions and reads the state of the blockchain (2) a validator that participates in consensus executes transactions, and keeps a full state of the ledger to service clients, or (3) a candidate validator that is an applicant to the validator position.

Initially, SRBB is bootstrapped with an initial set of validators pre-defined in the genesis block through a Know Your Customer (KYC) process[5]. Any node that wants to be a validator in the future (i.e., also known as a candidate validator) must first express interest by (1) depositing tokens (e.g. a pre-defined sum of ether – a unit of cryptocurrency used in Ethereum) in a reconfiguration smart contract or (2) authenticating using a KYC process. SRBB periodically elects validators to a committee from the set of candidate validators by calling an election. In this election, the current set of validators elect the next set of validators using a reconfiguration smart contract. The details of the validators of a committee are registered in the smart contract after an election outcome and each SRBB validator gets to know of other validators in the committee through an event emitted by the reconfiguration smart contract. The periodic rotation of SRBB validators mitigates the validator committee from being bribed by a slowly-adaptive adversary. Note that registering validator details and their stake in a smart contract provides a weaker form of anonymity in comparison to PoW blockchains like Bitcoin where anyone can join or leave the blockchain without the need to maintain a stable identity with a reserved stake.

The requirement to deposit tokens or pass a KYC process to be a candidate validator provides a form of Sybil resistance. The former makes it costly for a single user to assume the identities of multiple candidate validators, and the latter prevents a user from assuming multiple identities through authentication. We leave intricate details of the SRBB membership open-ended such that either depositing tokens or a KYC process can be used.

In this chapter, to produce a functioning reward-penalty mechanism (Section 3.2.5), we consider that depositing tokens is required to become a candidate validator. Note that a high deposit may impact, in theory, transaction fees. This is because one needs to set transaction fees high enough to exceed the payout from a Sybil-attacking coalition that is willing to spend enough deposits to seize control of the protocol. To alleviate this stress on the transaction fees, the validator deposit is recoverable after a locked period like in the design of PoS protocols [108].

3.2.3 TVPR (Transaction Validation and Propagation Reduction)

In TVPR, instead of validators eagerly validating and propagating transactions received from peer validators individually, validators only eagerly validate transactions received directly from clients. These validated transactions are then included in blocks and propagated. In other

[5] A process to uniquely identify an entity is who they claim to be by verifying identity information.

words, we remove step (3) in Figure 3.1 of the modern blockchain protocol where each validator individually propagates transactions to peers to produce Transaction Validation and Propagation Reduction (TVPR). When considering Alg. 1, we remove Alg. 1, line 10 to prevent validators from propagating transactions (i.e., broadcasting transactions) individually to peers. This way, validators do not require to eagerly validate and propagate transactions received from peer validators. To be more precise, in modern blockchains during the initial transaction propagation, if the number of validators is n, a transaction t is eagerly validated n times, whereas TVPR eagerly validates a transaction t once (i.e., only the validator receiving the transaction directly from the client eagerly validates it). Thus, we remove the redundant eager validation and propagation of transactions. Note that transactions are still included in blocks and propagated. In Section 3.3 we discuss in detail how reducing the eager validation of transactions does not cause the execution of invalid transactions. In Section 3.5.2, we present the throughput improvements and latency and transaction loss reductions of integrating TVPR to SRBB compared to modern blockchains. The potential drawbacks of TVPR are deferred to the discussion section (Section 3.6).

3.2.4 Transaction Life Cycle of SRBB

We now present the transaction life cycle of SRBB. An SRBB node consists of the SRBB VM and SRBB consensus. The SRBB VM is built upon Geth and integrates with TVPR. SRBB consensus uses the DBFT consensus protocol [36] combined with the superblock optimization of RBBC [12]. When describing the transaction life cycle of SRBB, we do not dwell deeply on the consensus as it is not our contribution[6]. Instead, we give sufficient information to understand SRBB in its entirety, highlighting our novelties. A transaction submitted by a client to SRBB goes through the stages below:

1. Reception: The client creates a properly signed transaction and sends it to at least one SRBB validator where the transaction is eagerly validated (Alg. 1, line 6). If the eager validation fails, the transaction is discarded. Otherwise, the transaction is kept in a pending queue in the transaction pool (Alg. 1, line 8). Once the transactions in the transaction pool pending queue reaches a threshold (pre-defined in the configurations), a validator creates a block b_i and adds it to a block queue (Alg. 1, line 13). Subsequently, the transactions used to create the block are removed from the transaction pool pending queue $\mathbb{P}$. Next, the validator fetches the first block from the block queue (Alg. 1, line 15) and propagates it to all validators (Alg. 1, line 16). Note that, unlike modern blockchains, SRBB does not propagate transactions individually. SRBB simply includes transactions in blocks and propagates blocks to the network, hence implementing our TVPR solution. In fact, SRBB is at the time of writing, the only blockchain with TVPR. For index k of the blockchain every correct SRBB validator propagates a block b_i s.t. i is the ID of the SRBB validator and $i \in \mathbb{Z}^+$. These blocks are propagated via reliable broadcast [109].

2. Consensus: An SRBB validator, after receiving a set of blocks $\mathbb{B}$ from peer validators via reliable broadcast [109] for index k of the blockchain where $\mathbb{B} \leftarrow \bigcup_{i=1}^{j} b_i$ s.t. $j \leq n$, executes

[6] we defer the details of the consensus to the original publication [12]

DBFT consensus [36] (Alg. 1, line 18) as outlined in Alg. 2, and decides a superblock $\mathbb{B}^*$. The SRBB validator after deciding the superblock sends it to the SRBB VM for execution (Alg. 1, lines 19-26). The decided blocks in the superblock are removed from the block queue (Alg. 1, line 27). All undecided blocks are kept in the block queue to be included in future consensus rounds. The consensus proceeds to the next round after a superblock is decided, and starts propagating a block again from the front of the block queue via reliable broadcast, if not already propagated (Alg. 1, lines 15-16).

Note that the only similarity between SRBB and RBBC [12] is this consensus phase. RBBC does not support the execution of smart contracts or DApps and does not have TVPR and RPM.

Algorithm 2 SRBB consensus

```
1: 𝔹 ← {b_1, b_2, ...b_i}                                   ▷ reliably delivered blocks
2: blocks ← ∅                                               ▷ blocks delivered from rbbroadcast are stored here in line 8
3: index ← 0                                                ▷ consensus round
4: blockQueue ← ∅                                           ▷ pending blocks to propose to consensus
5: consensus(𝔹):
6:    decCount ← 0, decBlocks ← 0
7:    for all block ∈ 𝔹 do
8:       blocks[i] ← block                                  ▷ add rb-broadcasted blocks to list s.t. i is ID of sender
9:       decBlocks[i] ← b-consensus-propose(i, true)        ▷ props. to binary cons.
10:   wait until ∃i : b-consensus-decide(i) is true         ▷ till binary cons. decided
11:   for j from 0 to n do
12:      if blocks[j] == ∅ then
13:         decBlocks[j] ← b-consensus-propose(j, false)
14:      if decBlocks[j] == true then decCount ← decCount + 1
15:   wait until decCount == n
16:   superblock ← ∅
17:   for i from 0 to n do
18:      if decBlocks[i] == true then
19:         superblock ← superblock ∪ blocks[i]             ▷ combine decided blocks
20:   return superblock
```

3. Commit: An SRBB VM, upon receiving the superblock $\mathbb{B}*$, takes a block b_i at a time from $\mathbb{B}*$, iterates through its transactions (Alg. 1, line 20), and attempts to execute them (Alg. 1, line 21). In the execution process, first, the SRBB VM lazily validates the transaction (Alg. 1, line 29). If a transaction's lazy validation succeeds, the SRBB VM attempts to apply the transaction to the current blockchain state (Alg. 1, line 32). A state transition for executing a transaction only occurs if the transaction is valid and non-conflicting. Since lazy validation is not as strict as eager validation (Chapter 2, sec. 2.1.10), a transaction may pass the lazy validation but still be invalid. The SRBB VM like modern blockchains handles such cases by throwing an error without transitioning state (Alg. 1, line 34). More specifically, during transaction execution, the SRBB VM rechecks other validity criteria not present in lazy validation (e.g., transaction signature verification, transaction size limit check), and throws an exception if invalidity is found.

If either the lazy validation fails or applying the transaction fails that means the transaction is invalid. The SRBB VM in this scenario discards the invalid transaction from the block b_i (Alg. 1, line 23) and moves on to the next transaction in the block. Subsequently, b_i is appended to the blockchain (Alg. 1, line 25) and the SRBB VM moves to the next block in the superblock. The SRBB VM follows the same procedure to process the subsequent blocks in the superblock until all the valid blocks are written to the blockchain.

Algorithm 3 The Reward-Penalty Mechanism

Initial State:
For block b: h_t, P_k and S_k are the hash of its txs, the block sender pub and priv keys resp.
$Cert_B \leftarrow \{P_k, (h_t)_{S_k}\}$ is the certificate of a block
T is the set of transactions in b where $t \in T$
r_b is a constant block reward
c is the cost of eager validating a transaction
count is $(h \rightarrow val)$ where *count* is a map between a hash and its count
N_B is the block number
i is the block index in the superblock and *round* is the consensus round

```
10: propReceived(Cert_B, T, i, round):                    ▷ validators invoke when block decided
11:   if invoked[hash(i, round)] == true then             ▷ already invoked for b in i, round
12:     exit                                              ▷ exit function
13:   invoked[hash(i, round)] = true
14:   P_k, (h_t)_{S_k} ← retrieve(Cert_B)                 ▷ retrieve data from Cert
15:   address_v ← derive(P_k)                             ▷ derive address of block proposer
16:   if address_v ∉ V then
17:     exit                                              ▷ invalid Cert_B, return
18:   else
19:     h_t ← P_k((h_t)_{S_k})                            ▷ retrieve hash of transactions from Cert
20:     if hash(T) == h_t then
21:       count[hash(P_k, T, i, r)] ← count[hash(P_k, T, i, r)] + 1   ▷ inc. count
22:       if count[hash(P_k, T, i, r)] == n-t then        ▷ thresh. decided the same block
23:         address ← derive(P_k)                         ▷ derive address of block proposer
24:         I ← r_b                                       ▷ define incentive
25:         C ← |T| · c                                   ▷ cost of eager validating transactions in b
26:         R ← I − C                                     ▷ calculate reward
27:         K[address] = K[address] + R                   ▷ add reward to proposer deposit
28:         count[hash(P_k, T)] = 0                       ▷ reset count
29: report(Cert_B, N_B, t, T):                            ▷ Report validator Cert. block number and transaction
30:   P_k, (h_t)_{S_k} ← retrieve(Cert_B)                 ▷ retrieve data from Cert
31:   address ← derive(P_k)
32:   h_t ← P_k((h_t)_{S_k})                              ▷ retrieve hash of transactions from Cert
33:   if address_v ∉ V or hash(T) ≠ h_t or t ∉ T then
34:     exit                                              ▷ invalid Cert_B or false report, return function
35:   else
36:     count[hash(P_k, N_B, t)] = count[hash(P_k, N_B, t)] + 1   ▷ inc. report count
37:     if count[hash(P_k, N_B, t)] = n-t then           ▷ thresh. val. reported invalid t
38:       address ← derive(P_k)                           ▷ derive validator that added invalid-tx
39:       K[address] = K[address] − P                     ▷ reduce byz. validators deposit by P
40:       for all v ∈ V and v ≠ address do                ▷ distribute penalty with validators
41:         K[v] = K[v] + P/(|V| − 1)
42:       emit address                                    ▷ emit byz. validator event
```

3.2.5 The Reward-Penalty Mechanism (RPM) for SRBB Validators

With modern blockchains and with SRBB, the invalid transaction propagation problem (Section 3.1.2) exists as Byzantine validators can propose invalid transactions in blocks that are propagated to the network. To cope with this problem we propose RPM as a part of SRBB.

While we previously distinguished correct from Byzantine validators, in the real world, validators behave rationally [110]. In other words, validators behave selfishly to maximize rewards instead of blindly following the protocol. Thus, it makes sense for rational validators to increase their gains by bypassing eager validation and proposing invalid transactions in blocks to save validation costs. RPM incentivizes rational validators to not propose invalid transactions within blocks and propagate such blocks. As a result, RPM mitigates transaction losses and performance degradation (Table 3.1). Like many blockchain reward mechanisms, RPM also rewards validators in a consensus round for proposing blocks. To the best of our knowledge, none of the reward and penalty mechanisms mitigate the propagation of invalid transactions within blocks [39, 111, 112, 113, 114].

The block proposal game: We use game theory to model the strategies of SRBB validators. We define a game G per consensus round as a tuple (V, S, U) where V is the set of players who are SRBB validators, S is the set of strategies followed by players and U is the pay-off (i.e., reward or penalty) for each strategy.

A validator could follow a correct strategy or a Byzantine strategy. We consider the correct strategy as a validator proposing valid blocks (i.e., blocks with valid transactions) by eagerly validating all transactions before including them in blocks. A Byzantine strategy is when a validator proposes invalid blocks by including invalid transactions in blocks (e.g., not eagerly validating transactions to save costs). Our goal is to design rewards for strategies so that the best strategy for a rational validator to follow is the correct strategy.

Reward-Penalty Mechanism: We present our RPM in Alg. 3. Earlier, we assumed out of n validators, at most f are Byzantine where $f < n/3$ (Section 3.2.1). As the committee progresses we assume that validators behave rationally.

Upon deciding on a superblock, validators invoke a propReceived function for each block in the decided superblock parsing the block proposer's certificate $Cert_B$, the set of transactions T in the block, the index of the block in the superblock i and the consensus round r (Alg. 3, line 10). The certificate of the block proposer $Cert_B$ consists of the public key of the block proposer P_k and the signed hash of the transactions in the block $(h_t)_{S_k}$ where h_t is the hash of transactions and S_k is the private key of the block proposer. Thus, $Cert_B = \{P_k, (h_t)_{S_k}\}$. Alg. 3, line 12 exits if a validator invokes propReceived more than once for the same i and r (i.e., prevents duplicate invocations). One validator cannot parse the same $Cert_B$ and T more than once to the propReceived function either as there is a checker preventing this, although not included in Alg. 3 for brevity.

For each invocation of propReceived by a distinct validator, RPM retrieves data from the certificate $Cert_B$ (Alg. 3, line 14) and checks the validity of $Cert_B$ by verifying whether the

$address_v$ derived from P_k is in the validator set of addresses V (Alg. 3, line 16). If not, $Cert_B$ is invalid and proposed by a non-validator so the propReceived function exits (Alg. 3, line 17). Otherwise, RPM retrieves h_t from $(h_t)_{S_k}$ (Alg. 3, line 19) and checks if the hash of T (i.e., hash(T)) is equal to h_t (Alg. 3, line 20), verifying whether the block proposer with P_k proposed a block with transactions T. If hashes are equal, Alg. 3 increments the propReceived invocation count for a block in a superblock for a particular P_k and T (Alg. 3, line 21). If at least $n - t$ validators have decided on a superblock with a block b that has the same T and P_k, RPM derives $address$ from P_k, calculates the reward from Alg 3, lines 24- 26 (i.e., Section 3.2.5 presents the reward design in detail) and increases the deposit of the proposer whose block is included in the decided superblock (Alg. 3, line 27). Finally, when the epoch ends, the tokens added to a validator's deposit exceeding D_v are funded to the validator's wallet address (not included here for brevity).

Invoking propReceived is in the best interest of a rational validator due to the following reason: If a validator v_1 eventually does not invoke propReceived for a decided block $b1$ proposed by validator v_2 then v_2 may also decide not to invoke propReceived for a decided block $b2$ proposed by v_1. This could result in both blocks proposed by v_1 and v_2 not reaching the $n - t$ threshold and receiving a reward accordingly (Alg. 3).

RPM penalizes rational validators that propose blocks including invalid transactions in the following way: upon noticing an invalid transaction in a block that is part of a decided superblock, validators become reporters invoking report in Alg. 3, line 29 parsing $Cert_B$ (certificate of the block proposer that included the invalid transaction), N_B (the block number containing the invalid transaction), t (the invalid transaction), and T (the set of transactions T in N_B). Then the validity of $Cert_B$ is verified and the function exits if an invalidity is noticed (Alg. 3, lines 30- 34) avoiding false reporting. Otherwise, a count is incremented corresponding to the number of validators that observed the invalid transaction t in block N_B proposed by the block proposer with public key P_k (Alg. 3, line 36). If at least $n - t$ validators have made the same report (Alg. 3, line 37), then the Byzantine validator that proposed a block with invalid transactions receives a penalty P s.t. $P = K[address]$, which is deducted from its deposit (Alg. 3, line 39) leaving the deposit at 0. The deducted penalty is then equally distributed among the other validators in the committee (Alg. 3, line 41). In RPM validators lose deposits only if they propose a block that includes invalid transactions (Alg. 3, line 39). Correct validators eagerly validate each transaction before inclusion in a block proposal. As such, correct validators do not lose their deposit. Finally, an event is emitted containing the wallet address of the Byzantine validator (Alg. 3, line 42). All correct validators listen to this event and exclude the Byzantine validator from future communications within the committee.

Alg 3 contains a block reward value r_b, an incentive I, an eager validation cost c, and a cumulative reward R for including a block in a superblock. The goal is to design these rewards in a way that the best strategy for a validator to follow is the correct strategy. Next, we present our reward design.

Reward design Let us consider R as the cumulative reward for proposing a block, I as the incentive, C as the cost of eager validating all transactions in a block, P as the amount

deducted in ether from a validator's deposit if the validator follows a Byzantine strategy (i.e., the entire current deposit is taken out), r_b as a constant ether value issued to validators for proposing a block, and $\sum_{n=0}^{N_{tx}} Txfees$ as the total transaction fees in the proposed block (i.e., the transaction fee is not included in RPM in Alg. 3 as it is charged separately at the SRBB VM when transactions are executed). Then the two reward equations are: $R = I - C - P$ and $I = r_b + \sum_{n=0}^{N_{tx}} Txfees$. We do not include double spending rewards rational validators may gain as double spending is mitigated through committee reconfiguration.

From Theorem 2, proposing invalid transactions results in a validator losing her entire deposit. Rational validators try to maximize their gains effectively. Therefore, RPM discourages proposing invalid transactions in blocks.

3.3 Smart Redbelly Blockchain: Implementation

In this section, we present design choices and implementation details of some important aspects of SRBB that helped it become a highly performing blockchain.

3.3.1 Consensus Implementation

The SRBB implements the DBFT consensus protocol in Golang to achieve compatibility with the SRBB VM ported from Geth (i.e., the Golang implementation of the EVM). More specifically, we used the Golang version 16.2.2, the newest version at the time of developing SRBB. The SRBB DBFT consensus includes a number of optimizations.

Batching messages: Batching messages is a classic optimization in distributed systems. In our Golang implementation of DBFT, we used Golang RPC calls to communicate between validators. When there were multiple messages ready to transmit through the wire on a validator node to the same destination validator, we combined the messages into a batch and send the requests in a single RPC. This batching optimization helped scale the Golang implementation of DBFT to 200 geo-distributed VMs on AWS spanning 5 continents (Section 3.5).

Choice of RPC library and encoding method: We used the Golang *net/rpc* library with gob (golang binary) encoding to execute RPC calls. We chose *net/rpc* with gob encoding because it yielded better performance compared to Golang gRPC.

Keep-alive connections: At bootstrap time of SRBB, we established keep-alive connections between validators. Thus, every time an RPC call needed to be made between validators for communication, a connection did not need to be established, reducing the communication latency.

Hashing messages: A hashing optimization is included in the reliable broadcast similar to Redbelly consensus [36] to reduce the size of the broadcasted messages. More specifically, a SHA256 hash is derived for each block and a mapping between the block and the hash value is

kept. The block is then propagated to validators. A reliable broadcast is subsequently started for the hash derived from the block. When the hash is reliably delivered, the corresponding block in the mapping is considered reliably delivered. Next, a binary consensus instance starts for each reliably delivered block. The use of hashes benefits the performance in two ways. First, it reduces the usage of the network bandwidth. Second, it reduces the CPU and memory usage at a validator when encoding and decoding messages during wire transfers.

3.3.2 SRBB VM implementation

The SRBB VM was ported from the Geth version v1.10.18, the latest at the time of development, and changed to prevent redundant eager validation and propagation of transactions by integrating TVPR. Below we mention in detail our implementation of TVPR on Geth to produce SRBB VM. We also outline that despite Geth's recent releases, our TVPR is still relevant.

TVPR implementation: We integrated TVPR into the EVM by disabling the initial individual transaction propagation among validators. This way only the first SRBB node receiving transactions from clients eagerly validated and included the transactions in blocks and propagated them to the network. Implementing TVPR involved changing the Geth implementation, which required a deep dive into the implementation details of Ethereum and discussions with Ethereum core developers. In total, we changed 161 LOC (Lines of Code) for the TVPR implementation. These changes included: (1) disabling the event that notifies successful eager validation of each transaction to the function that broadcasts transactions individually and (2) disabling functions that handle individual transactions received from peers.

One might think that SRBB allows the execution of invalid transactions because a transaction is eagerly validated only once by a SRBB validator and then lazily validated at each SRBB validator prior to execution, where the lazy validation is not as strict as the eager validation. Note that the reduction of transaction eager validations does not cause the execution of invalid transactions in SRBB. Instead, in the case of invalid transactions, the SRBB execution throws an exception. More precisely, a transaction is valid only if (i) the transaction is properly signed, (ii) its size does not exceed a limit, (iii) its nonce is the next sequence number, (iv) its gas cost is covered by the sender balance, (v) its transferred amount is covered by the sender balance. The lazy validation checks (iii), (iv), (v) whereas the execution checks any remaining invalidity. In particular, the Geth implementation[7] which SRBB builds upon, raises an *ErrInvalidSig* exception if (i) is not satisfied. Overflow and VM exceptions are raised if (ii) is not satisfied[8].

Note that since we developed SRBB VM, the official implementation of Geth has changed. To the best of our knowledge, none of these changes have impacted the relevance or the guarantees of TVPR. This is because after reducing eager validations, if an invalid transaction bypasses lazy validation, the execution will attempt to execute the invalid transaction and throw an exception if any remaining invalidity is found, without transitioning state.

[7] L635 of https://github.com/ethereum/go-ethereum/blob/master/core/types/transaction.go of commit c4a6621

[8] L187-219 of https://github.com/ethereum/go-ethereum/blob/master/core/vm/interpreter.go, and https://github.com/ethereum/go-ethereum/blob/master/core/vm/errors.go.

Superblock compatible with SRBB VM: As the SRBB consensus returns a superblock, and since the superblock increases in size with the number of nodes, the SRBB VM slows down for large n consuming excess CPU, and memory having to process large superblocks. As a solution, we optimized the SRBB VM to fully process one proposed block (i.e. sub-block) of the superblock at a time allowing it to alternate frequently between CPU-intensive (verifying signatures and transaction executions) and memory-intensive (block writes) tasks reducing the stress on resources.

3.4 Smart Redbelly Blockchain: Proofs of Correctness

Here we present the proofs of correctness of SRBB. First, we prove that SRBB solves the blockchain consensus problem (Chapter 2, Section 2.3.2). Second, we prove that RPM mitigates rational validators from proposing invalid transactions in blocks.

Theorem 1. *SRBB solves the blockchain consensus problem.*

Proof. We prove that each property of Def. 1 is preserved by SRBB.

Liveness: As a result of removing line 10 of Alg. 1 in SRBB, transactions are no longer propagated individually to the network among validator peers and eagerly validated at each SRBB validator. However, correct validators still create blocks including valid transactions, and propagate them to peers (Alg. 1, line 16). Thus, every correct SRBB validator receives a set of blocks $\mathbb{B}$ propagated by correct SRBB validators at index k (Alg. 1, line 17). An SRBB VM decides a set of blocks $\mathbb{B}^*$ s.t. $\mathbb{B}^* \subseteq \mathbb{B}$ (a superblock) and stores the valid transactions in these decided blocks on the blockchain (Alg. 1, lines 18-25). While the decided blocks are removed from the block queue (Alg. 1, line 27), undecided blocks are kept in the SRBB block queue to be included in future blocks. These transactions are eventually re-included in a future decided block by correct SRBB validators and stored in the blockchain. Thus, every transaction received by a correct SRBB validator is eventually stored in the block sequence of all correct SRBB validators.

Safety: The preservation of safety is proved by contradiction. If none of the two chains of blocks maintained locally by any two correct SRBB validators $v1$ and $v2$ is a prefix of one another, it means the superblock $\mathbb{B}^*$ decided at index k (Alg. 1, line 18) of $v1$ and the superblock $\mathbb{B}^{*\prime}$ decided at index k of $v2$ are different. This results in two different transaction executions (Alg. 1, line 21) for $v1$ and $v2$. However, this is a contradiction because any two correct SRBB validators $v1$ and $v2$ should decide on the same superblock at index k due to consensus guarantees of DBFT [36] (Alg. 1, line 18). Thus, any two validators $v1$ and $v2$ should store the same block b at index k of the chain (Alg. 1, line 25). Therefore, any two correct validators should maintain locally an identical chain of blocks or a chain where one is a prefix of another (i.e., because blocks do not get decided, executed, and stored at the same time in all SRBB validators) resulting in the same execution. Thus, through proof by contradiction, SRBB achieves safety.

Validity: Due to the consensus protocol, all correct SRBB validators decide on the same

superblock $\mathbb{B}^*$ at index k (Alg. 1, line 18). If each block b_i in the superblock $\mathbb{B}^*$ has a valid set of transactions, it is appended to the blockchain (Alg. 1, lines 20-25). Thus, each block appended to the blockchain of each correct SRBB validator is a set of valid non-conflicting transactions.

□

Theorem 2. *Rational validators proposing invalid transactions in blocks lose their deposit.*

Proof. If a rational validator proposes invalid transactions in a block to minimize C and thereby increase R (i.e., $R = I - C$), other validators are incentivized to report the invalid transaction along with the proposing validator (Alg. 3 lines 29- 41). As a result, a penalty P is deducted from the reported validator's deposit, and the validator is removed from the committee (Alg. 3, lines 39 and 42). When a Byzantine validator's block with invalid transactions is decided, we know that they gain a reward of $I - C'$ where $C' < C$ (i.e., Byzantine validators skip eager validation to save cost). If the initial deposit of the Byzantine validator is D, now it becomes $D' = D + I - C'$. However, once reported by $n - t$ validators, Byzantine validators lose P s.t. $P = D' = D + I - C'$. Thus, the Byzantine validator's deposit becomes $D_{end} = D + I - C' - P$ leading to $D_{end} = 0$ eventually. Thus, the Byzantine validator loses their entire deposit D eventually. □

3.5 Smart Redbelly Blockchain: Evaluation

In this section, we present our evaluation of SRBB and its performance compared to 6 modern blockchains. We show that for realistic DIABLO DApp workloads of NASDAQ, Uber, and FIFA, SRBB outperforms Algorand, Avalanche, Diem, Ethereum, Quorum, and Solana. In other words, SRBB shows higher throughput, lower latency, and fewer transaction losses compared to 6 modern blockchains under the evaluated realistic DApp workloads. We also show that SRBB outperforms a naive blockchain comprising of the EVM, the DBFT consensus, and the superblock optimization. This goes to show that our TVPR and RPM contributions improve SRBB's ability to support realistic DApp workloads.

3.5.1 Experimental Setting

We evaluated SRBB using the DIABLO blockchain benchmark suite [28] that evaluates blockchains against specified workloads by sending pre-signed transactions. We used the realistic DApp workloads of NASDAQ, Uber, and FIFA that span 3, 2 and 3 minutes respectively[9]. NASDAQ (peak request rate - 19800 TPS, avg. request rate - 168 TPS) uses a real trace of Apple, Amazon, Facebook, Microsoft, and Google stock trades executed on a DApp, Uber (peak request rate - 900 TPS, avg. request rate - 852 TPS) uses a real trace from the mobility service Uber executed on a DApp and FIFA (peak request rate - 5305 TPS, avg. request rate - 3483 TPS) uses a real workload from the soccer world cup executed on a DApp. For the DApp workload experiments, we used 200 validators spanning 10 AWS regions (i.e., 5 continents), namely: Bahrain,

[9] https://github.com/lebdron/minion/tree/aec

Cape Town, Milan, Mumbai, Sao Paulo, Ohio, Oregon, Stockholm, Sydney, and Tokyo. For all DApp benchmarks, we used the same AWS c5.2xlarge EC2 instances (8 vCPUs, 16 GB RAM – equivalent to a modern-day PC) as DIABLO [28].

Rationale for machine selection: The c5.2xlarge AWS instance type was consistently used throughout all SRBB benchmarks for two reasons. First, to make all results comparable with DIABLO [28]. Second, to make the evaluations encompass a wide range of blockchains as some blockchains require specific CPU and memory requirements (e.g., Solana) that cannot be met with smaller AWS instances.

Rationale for selecting 6 modern blockchains to compare with SRBB: Evaluating all blockchains in existence against SRBB is not realistic. We used the 6 modern blockchains evaluated in DIABLO for comparison against SRBB because DIABLO reported a thorough evaluation of these blockchains under realistic DApp workloads. To make the comparison fair, we used the same configuration parameters as used in DIABLO [28] when evaluating SRBB.

Performance metrics: Our evaluation focuses on the throughput, latency, and transaction loss of blockchains which are indicators of blockchain congestion. With more congestion, a blockchain's throughput drops, and latency and transaction losses increase.

In summary, all experimental parameters were the same as the ones used in DIABLO [28] for realistic DApp evaluation. Similar to the DIABLO DApp evaluations, all workloads were evaluated with one full experimental iteration. This is because our discussions and comparisons with the authors of DIABLO revealed multiple runs of the same DIABLO DApp workload experiments yield minimal statistical variance in the results due to a long experimental time of ∼5 minutes (i.e., DApp workloads send transactions for 2-3 minutes and blockchains typically processed these transactions for ∼5 minutes).

SRBB reached a maximum average throughput of ~2000 TPS for realistic DApp workloads. SRBB achieved higher throughput and fewer transaction losses compared to 6 modern blockchains for the realistic DApp workloads of NASDAQ, Uber, and FIFA [28].

Moreover, SRBB was the only blockchain out of the evaluated blockchains to not lose transactions for the realistic DApp workloads of NASDAQ and Uber. For the demanding FIFA workload, SRBB committed over 98% of transactions. The higher throughput, lower latency, and fewer transaction losses of SRBB compared to modern blockchains indicate that SRBB reduces blockchain congestion.

For the remainder of this section, first, we compare SRBB with modern blockchains (Section 3.5.2) for the real DApp workloads of NASDAQ, Uber, and FIFA from the DIABLO blockchain benchmarking suite. Then we evaluate the benefits in transaction losses and performance of RPM in SRBB when Byzantine validators propagate invalid transactions.

3.5.2 Comparison with Other Blockchains

Figures 3.2 and 3.3 depict the performance of 6 modern blockchains (i.e., Algorand, Avalanche, Libra-Diem, Ethereum Proof-of-Authority, Quorum IBFT, Solana) compared to SRBB and EVM+DBFT which is a naive smart contract supported version of RBBC that combines the Ethereum VM with the superblock optimized DBFT consensus but does not have TVPR and

RPM. The evaluation of EVM+DBFT is included to show that the performance benefits of SRBB come from TVPR and not from the prior works of the superblock optimization and the DBFT consensus of RBBC.

Note, some blockchains did not yield an average latency or throughput value for certain workloads (e.g., 0 TPS and 0 s latency) because the transaction costs exceeded their budget [28] or they crashed due to the high load.

Figure 3.2 presents the average throughput in the y-axis and transaction commit percentage as a value at the top of the bar for the NASDAQ, Uber, and FIFA workloads, depicted by (N,U,F) in Figure 3.2 for brevity. SRBB is the only blockchain to commit 100% of the transactions for the NASDAQ and Uber workloads. SRBB also commits 98% of transactions for the demanding FIFA workload where no other blockchain commits more than 47% of transactions. SRBB reaches average throughputs of 167 TPS, 828 TPS, and 1808 TPS for the NASDAQ, Uber, and FIFA workloads respectively, which are the highest average throughputs for all evaluated blockchains. Figure 3.3 shows the average latencies for the (N,U,F) workloads. SRBB yields the least average latency among all evaluated blockchains for both the NASDAQ and Uber workloads with average latencies of 6.6 and 3.9 seconds respectively. For the FIFA workload, SRBB yields an average latency of 64 seconds. This slightly higher average latency of SRBB over Avalanche, Diem, and Solana in the FIFA workload is due to SRBB committing 98% of the transactions as opposed to only 2% or fewer transaction commits in the other blockchains. All 6 modern blockchains yield throughputs lower than 900 TPS and latencies higher than 20 seconds. These performances are much lower compared to their claimed performances [28], indicating major performance degradation.

Most importantly, SRBB multiplies the average throughput by 55×, divides the latency by 3.5, and reduces transaction losses considerably compared to EVM+DBFT. Since the difference between SRBB and EVM+DBFT is TVPR, TVPR is responsible for better performance and reduction in transaction losses and not prior works from RBBC [36, 12].

The disparity between the performance of EVM+DBFT and SRBB is due to the following reason: Executing the superblock and validating and propagating transactions redundantly is expensive on the EVM. This is why naïve EVM+DBFT shows weak performance. However, with the TVPR optimization, the validation and propagation of transactions redundantly is reduced. This reduces the heavy processing required by the EVM from two computationally expensive tasks to just one. Thus, SRBB performs significantly better than EVM+DBFT.

3.5.3 Evaluation of Performance with Byzantine Validators

Here we present the performance of SRBB when a Byzantine validator propagates invalid transactions. We created invalid transactions by setting the balance of the transaction sender to 0 ETH. More specifically, we compare the average throughput and transaction losses of SRBB without RPM and SRBB with RPM. Thus, this evaluation shows the performance benefits RPM yields considering throughput and transaction losses. Due to budget constraints, we had to restrict this benchmark to a single AWS region and the smallest validator size a BFT blockchain

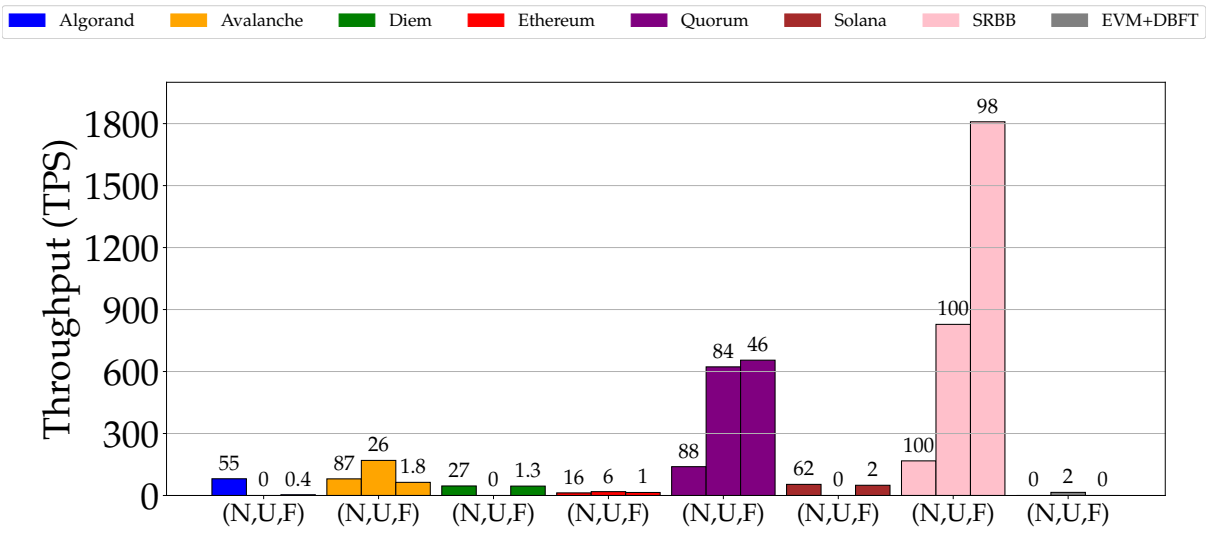

Figure 3.2: Throughput (y-axis) and commit percentage (top of the bar) for NASDAQ, Uber and FIFA workloads (i.e., (N,U,F) is NASDAQ, Uber and FIFA)

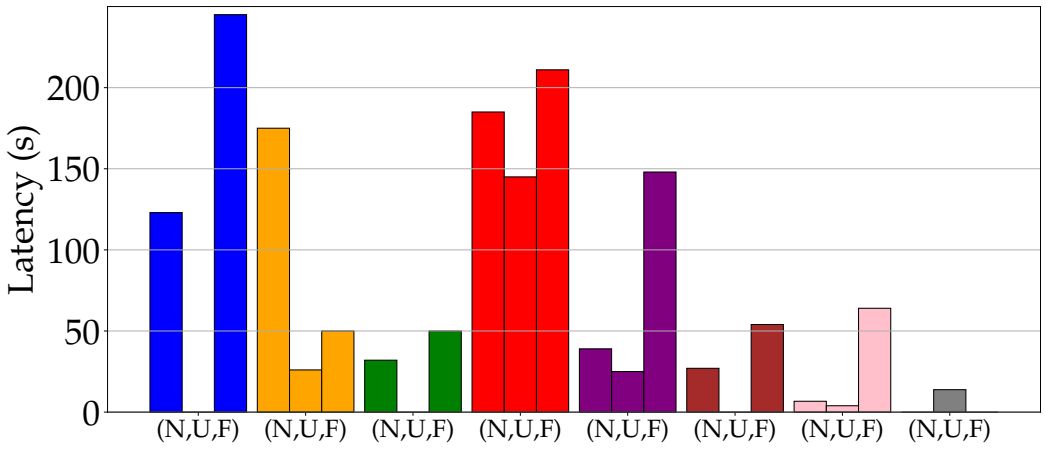

Figure 3.3: Latency (y-axis) for NASDAQ, Uber and FIFA workloads (i.e., (N,U,F) is NASDAQ, Uber and FIFA)

can tolerate[10] (i.e., four validators). More specifically, we performed this benchmark on the Sydney AWS region with 3 correct and 1 Byzantine validator.

	#valid txs sent	#invalid txs sent	#Byzantine Validators	throughput (TPS)	#valid txs dropped
SRBB w/o RPM	20K	10K	1	3998.2 TPS	none
SRBB w/ RPM	20K	10K	1	4285.71 TPS	none

Table 3.1: The average throughput and valid transaction drops of four SRBB validators where one is Byzantine.

For Table 3.1, we used a sending rate of 15000 TPS to stress test the blockchains. From Table 3.1, it is clear that despite Byzantine validators propagating invalid transactions in blocks, SRBB did not drop any valid transactions. RPM integrated with SRBB showed the best performance by increasing the average throughput to 4285 TPS which was 7% higher than SRBB without RPM. This is because Byzantine validators who are also rational (i.e., try to maximize their reward) do not propagate invalid transactions in the presence of RPM due to the fear of being penalized. Thus, RPM helps improve performance.

3.6 Discussion

In this section, we discuss the potential drawbacks of TVPR in SRBB and how they could be addressed. We also discuss the applicability of TVPR to modern blockchains.

Censorship of transactions: In modern blockchains (e.g., Ethereum) if a validator decides not to include a transaction in its new block, the said transaction is likely to be included in another block by another validator eventually due to transactions being propagated to all validators initially. This prevents censorship. With SRBB, since there is no individual transaction propagation among validators, if a validator decides not to include a transaction from a client in its new block, the transaction becomes censored (i.e., note we consider a validator that censors transactions as a Byzantine validator). One solution would be to let a distributed load balancer handle the censorship problem. A load balancer existing between clients and validators can randomly forward each transaction from a client to a different SRBB validator, to increase the probability of transaction inclusion in blocks. Even then, a transaction may be censored if the load balancer forwards the transaction to a validator that censors it. In this case, if the client does not receive a transaction receipt as proof of its execution within a period, the transaction can be resent by the client and forwarded to a different validator than before by the load balancer due to the randomness in forwarding. The new validator may not censor the transactions as the previous validator. If they also censor the transaction, this entire process can be repeated until a validator that does not censor the transaction receives it. This process can be automated

[10] Using many medium-sized instances instead of the four c5.2xlarge AWS instances used in Table 3.1 does not overcome our budgetary constraints as medium-sized AWS instances yield minimal savings compared to c5 instances.

to reduce the burden on the client. A Byzantine load balancer itself may be problematic. In such a case, a client may require to distribute transactions to multiple load balancers. We look at a few transaction load balancing and transaction distribution strategies among multiple load balancers to mitigate censorship in our future work [115].

Applicability of TVPR to other blockchains: In contrast to SRBB, implementing TVPR on modern blockchains can be problematic. Due to removing propagation of transactions among validators through TVPR, the first validator receiving a transaction should include it in a block for the transaction to be eventually executed. If the first validator receiving a transaction is weak, (e.g., has a low probability of creating blocks), it will rarely win the consensus protocol. Thus, a client can expect to wait a long time for their transaction to be included in a block. To prevent this drawback, clients may submit transactions to the most powerful validators in the hope of increasing the probability of their transactions being included in blocks sooner. This can centralize the blockchain towards a few validators and these few validators can be overloaded with transactions leading to a DoS.

In contrast, with SRBB, despite having TVPR, clients do not have to wait a long time for their transactions to be included in a block. This is because, in SRBB, all validators regardless of being weak or powerful can make block proposals per consensus round and combine their blocks to create a superblock [12]. More specifically, since SRBB uses the Redbelly consensus [12] (Section 3.2.4) there is no single validator winning one consensus round. Every validator gets to include a block in the decided superblock per consensus round if every validator proposes a block. Thus, a transaction sent by a client to any validator will be included in the superblock in the same consensus iteration or the next (e.g., with 1000 validators, a client does not have to wait for 1000 iteration before its transaction is included in a block as all 1000 validators can propose blocks per consensus round and include their block in the decided superblock).

Removing eager validations completely: Several works like Hyperledger Fabric [33] and PRISM [32] completely remove eagerly validating transactions. However, unlike SRBB, these two blockchains do not remove the initial transaction propagation as in our TVPR solution. The complete removal of transaction eager validations increases the chances of DoS attacks on the blockchain through spam transactions consuming network bandwidth. This is because there is no validator initially validating the transactions, leaving the potential for spam transactions to be submitted to the system. In contrast, SRBB's validators eagerly validate transactions received directly from clients. SRBB's RPM goes further and reduces the potential of Byzantine validators submitting invalid transactions to the network.

3.7 Summary

In this chapter, we introduced SRBB, a provably correct permissionless blockchain that improves performance in realistic DApps by mitigating blockchain congestion. We demonstrated that reducing transaction validations and propagation in normal cases using TVPR and mitigating invalid transaction propagation under flooding attacks using RPM can lead to minimal transaction losses and significant performance improvements in SRBB. These improvements

make SRBB perform significantly better than Algorand, Avalanche, Diem, Ethereum, Quorum, and Solana when executing DApps under real workloads.

Despite SRBB's better DApp support over state-of-the-art, it is unlikely to support a large number of DApps. In a realistic setting, a blockchain might receive an aggregation of multiple DApp workloads. For instance, if an aggregation of the NASDAQ, Uber, and FIFA workloads were sent to SRBB, there will likely be an increase in transaction losses and a decline in performance due to increased blockchain congestion. Therefore, in the next chapter, we present a dynamic transparent DApp-oriented sharding protocol that enables DApps to be executed concurrently in separate shards to reduce blockchain congestion and improve blockchain performance.

Chapter 4

DApp-oriented Dynamic Transparent Blockchain Sharding for Concurrent Execution of DApps

In this chapter, we propose a new dynamic transparent blockchain sharding protocol that improves DApp performance by concurrently executing different DApps on different shards, a concept known as DApp-oriented or service-oriented sharding [40]. We implement our sharding protocol on SRBB to empirically show the boost in DApp performance.

Previously (Chapter 3), we introduced SRBB to improve DApp performance. However, to support a large eco-system of millions of different DApps to widen the adoption of Web3, one needs to eventually rival the performance of the current centralized applications on the web. The great success of sharding in database performance led to the application of such techniques on blockchains [31, 30, 116]. The use of shards allows transactions to be executed concurrently in separate environments, reducing blockchain congestion.

Unfortunately, the existing blockchain sharding protocols (Table 4.1) suffer from limitations. In fact, they are typically *static*: once the blockchain is spawned, there is no way to change the number of shards at runtime based on DApp demand [54, 57]. Respawning the blockchain constantly to adjust the number of shards based on DApp demand is not ideal as blockchains are intended to run for a long time (e.g., Bitcoin [13] has been running for more than a decade without interruption). The popularity of these DApps is heterogeneous, and a new popular DApp may severely increase the number of requests to a particular smart contract, similar to the CryptoKitties DApp that congested the Ethereum network [122]. Ideally, a sharding protocol should allow the blockchain governance to resize the shards and adjust the shard number on-demand without *hard forking*, i.e., creating a duplicated instance of the same blockchain. This would reduce blockchain congestion and improve performance by provisioning more resources for popular DApps, grouping less demanded DApps on fewer shards, or offering more resources (e.g., CPU, storage) to a particularly congested shard.

Another problem is that most sharding protocols are *opaque* (cf. 2^{nd} column of Table 4.1): there is no way to securely access their shard configuration by querying the blockchain. Even if

	Sharded blockchains	Transparency	Dynamism	Shard number dynamism	Shard size dynamism	No synchrony needed
Payments	Elastico [71]	✗	◔	○	◑	✗
	OmniLedger [31]	✗	◑	○	●	✗
	RapidChain [30]	✗	◑	○	●	✗
	SSChain [84]	✗	◑	○	●	✗
	SharPer [83]	✗	○	○	○	✓
	Cerberus [117]	✗	○	○	○	✓
DApps	Avalanche [37]	✗	●	●	●	✗
	ChainSpace [81]	✓	◑	○	●	✓
	Cosmos [118]	✗	◕	●	◑	✓
	Ethereum sharding [116]	–	◔	○	◑	✓
	Polkadot [54]	✗	○	○	○	✓
	Zilliqa [57]	✗	○	○	○	✗
	SkyChain [119]	✗	◕	●	◑	✓
	Monoxide [45]	✗	○	○	○	✓
	AHL [120]	✗	○	○	○	✓
	RingBFT [121]	✗	○	○	○	✓
	Sharded SRBB	✓	●	●	●	✓

Table 4.1: Comparison of sharded blockchains: the dynamism ranges from low, as indicated by ○, to high, as indicated by ●, a checkmark ✓ indicates that the property holds while a cross ✗ indicates that the property does not hold. The first column represents the transparency of the sharding solution which refers to the ability for anyone to retrieve the sharding configuration from the ledger. The second column represents the dynamism of the sharding approach which is a combination of shard number dynamism (i.e., the ability to change the number of shards at runtime) and the shard size dynamism (i.e., the ability to change the size of a shard at runtime). The last column "No synchrony needed" refers to whether the sharding solution assumes synchrony or not.

a sharding protocol was made dynamic by offering the users the ability to change the number of shards at runtime, there would be no secure way for these users to confirm the changes took effect. In some cases, sharded blockchains offer a website where users can find information about the current shard configuration. For example, Cosmos [118] offers a website to observe a map of its zones [123]. However, such a web service is typically centralized and prone to a single point of failure, hence defeating the purpose of using a distributed ledger for security. First, this website could simply be hacked, conveying a misleading sharding configuration. Second, the traffic towards the website could be easily redirected with a network attack [48]. Finally, users could expose themselves to phishing attacks by accessing a hacked copy of the website instead of the real one. Such attacks are becoming frequent to fool blockchain users about the

information they access online [124]. These types of limitations can be avoided by recording the sharding configuration on the ledger, which allows anyone to access it transparently and securely.

We propose a novel DApp-oriented dynamic transparent blockchain sharding protocol that can concurrently execute DApps on separate shards to enhance performance while being able to dynamically adjust shards according to DApp demand. We implement our sharding protocol on SRBB. Since our protocol is intended to operate in open networks, it does not assume synchrony, but rather partial synchrony [47], in that the bound on the message delays is unknown (Chapter 2, Section 2.2). This protocol is made *transparent* by exploiting the blockchain itself: validators can (i) create, (ii) close or (iii) adjust the size of a shard by invoking functions of a smart contract residing on the default shard (called *mainchain*) within a limited time window (if the network asynchrony prevents them from succeeding, then they retry with a larger time window until success). As all smart contract invocations are logged to the secure storage of the distributed ledger, the shard configuration is securely visible from the world state making the sharding transparent. A new shard is created as a *shardchain* provisioned by the assets deposited on the mainchain by its users. An important challenge we had to solve was for the network topology to adapt based on the output of the sharding smart contract. To achieve this we present a reconfiguration function in the smart contract which emits an event that triggers the spawning, shutdown, and restart of some of the blockchain machines. Like other sharding approaches, we provide randomness in shard creation to prevent shard takeovers by malicious nodes. We implement and evaluate the ability of our sharding solution to execute DApps concurrently on SRBB [1]. Our results confirm that our sharding protocol leads to speedup, that the performance of shards can benefit from a growing number of node resources, and that our mainchain does not act as a performance bottleneck.

In summary, this chapter presents the following contributions:

- We introduce DApp-oriented dynamic transparent blockchain sharding. This sharding mechanism follows a service-oriented sharding model [40] where each shard is dedicated to a DApp or group of DApps disjoint from DApps executing on other shards. Each DApp executes on at most one shard. Our sharding mechanism also provides the ability for a blockchain to reconfigure the number of its shards and the size of each of its shards transparently and on demand without disrupting the blockchain service. The ability to dynamically shard based on DApp demand is particularly appealing to reduce blockchain congestion and thereby improve the performance of DApp execution on blockchains.

- We propose a DApp-oriented dynamic transparent sharding solution that (1) concurrently executes DApps in shards, and (2) provides the capability to create a new shard, adjust a shard, close a shard, and rotate shard validators. We present our sharding solution algorithms as inherently transparent smart contracts that emit events to replace the current sharding configuration at the network level.

- We demonstrate the feasibility of our approach by implementing our sharding algorithms on SRBB deployed in 10 countries across all 5 continents. The experimental results confirm

that the performance improves when shards execute DApps concurrently (Figure 4.2). We demonstrate the speedup of DApp execution when 3 shards of SRBB concurrently execute 3 DApp workloads separately [28]: the first shard executes NASDAQ, the second shard executes Uber and the third shard executes FIFA. We show that with just 3 shards executing the aforementioned demanding DApp workloads, a total average throughput of 2828.87 TPS, and a peak throughput of 9523 TPS can be achieved. In contrast, the average throughput and the peak throughput were halved when SRBB received an aggregate workload consisting of NASDAQ, Uber, and FIFA workloads.

Chapter Outline: The rest of this chapter is ordered as follows: In Section 4.1, we present our DApp-oriented dynamic transparent sharding protocol. Section 4.2 discusses the availability of our sharding solution and Section 4.3, proves our sharding solution is secure with high probability. In Section 4.4, we illustrate the performance of our sharding protocol when executing concurrent DApps deployed at a large scale on SRBB. Finally, in Section 4.5, we conclude the chapter.

4.1 DApp-oriented Dynamic Transparent Sharding Protocol

4.1.1 System Model

From n validators, a group of validators is selected through a random mechanism to the mainchain. The set of nodes that execute consensus on the mainchain is termed the mainchain committee. The mainchain committee is tasked with administrative tasks of the network such as shard creation, shard committee rotation, and dynamic adjustment of the number of shards and the nodes in a shard. A mainchain can create one or many shards from validators. Each shard keeps a separate state, and transactions and is tasked with executing a unique DApp or a set of related DApps. A validator wishing to join a shard should stake a deposit and register a wallet address prior to joining a shard to mitigate Sybil attacks.

4.1.2 Threat Model

To implement our sharding protocol, we alter a previous assumption of our threat model (Chapter 3, Section 3.2.1). More specifically, out of n validators, we assume f is Byzantine such that $f < n/4$. This is to ensure that when validators are split into shards, a shard has N_v validators such that $f_{N_v} < N_v/3$ with high probability (proofs are deferred to Section 4.3), similar to previous work [31] [71]. For a system executing BFT consensus within a shard, as is the case in our evaluations, $f_{N_v} < N_v/3$ ensures all correct validators agree. Note that n and N_v can vary at run-time due to the dynamism of our approach. A *shard-takeover attack* consists of an adversary gaining control over sufficiently many validators within a shard to prevent consensus from being reached. As mentioned previously (Chapter 3), we assume a slowly-adaptive adversary that can bribe validators between epochs but not when a shard committee is active [31] [30], and we cope with bribery by rotating shards periodically. More precisely, the shard committee (i.e. the

set of validators in the shard) rotates per epoch, mitigating bribery through a slowly-adaptive adversary.

4.1.3 Network Model

The sharding network model assumes honest validators in the network are well-connected and the communication channels between honest validators are partially synchronous (Chapter 2, Section 2.2). While various sharded blockchains typically assume a stronger property, called synchrony [47], synchrony is typically difficult to guarantee and can be violated in an open network like the Internet [48], which has led to numerous double spending attacks against blockchains [85, 86].

4.1.4 Bootstrapping

A subset of n validators are selected to perform consensus on the mainchain based on a random beacon. The mainchain committee rotates periodically to mitigate bribery takeovers and the size of the mainchain is changeable in a similar approach to how we change the shard size (Alg. 4 line 16), which allows new nodes to join the mainchain committee if a threshold of mainchain validators agree.

4.1.5 Overview

In this section, we present an overview of how the DApp-oriented dynamic transparent sharding protocol works.

Figure 4.1 depicts a high-level example of a dynamic blockchain sharding execution where smart contract invocations stored in blocks configure the sharding. Initially, there are 25 validators in the mainchain with a single genesis block, as depicted on the 1^{st} column, they decide the shard size. Then, other validators invoke the Join$(\cdot)$ function to register interest to join a new shard (cf. 2^{nd} column). When enough validators have cast their interest to join, the shard creation smart contract invokes CreateShard$(\cdot)$ on the mainchain to create a new (blue) shard (cf. 3^{th} column). The resulting function invocation is stored as a transaction in a new block on the mainchain. A new (blue) shardchain, maintained by the (blue) shard is created: it is linked to the block of the mainchain where its creation transaction is stored. New validators invoke the Join() function as depicted in the middle column. After that, the CreateShard$(\cdot)$ function creates a new (green) shard while the old (blue) shard invokes the CloseShard$(\cdot)$ function, which reports the blue shard history to a new block on the mainchain (cf. 4^{th} column). Finally, the new shard rotates its participants by executing the Replace$(\cdot)$ function whose invocation gets stored in a new block.

In the next sections, we explain how the DApp-oriented dynamic transparent sharding (1) executes concurrent DApps in shards (DApp-oriented sharding) and (2) achieves transparency.

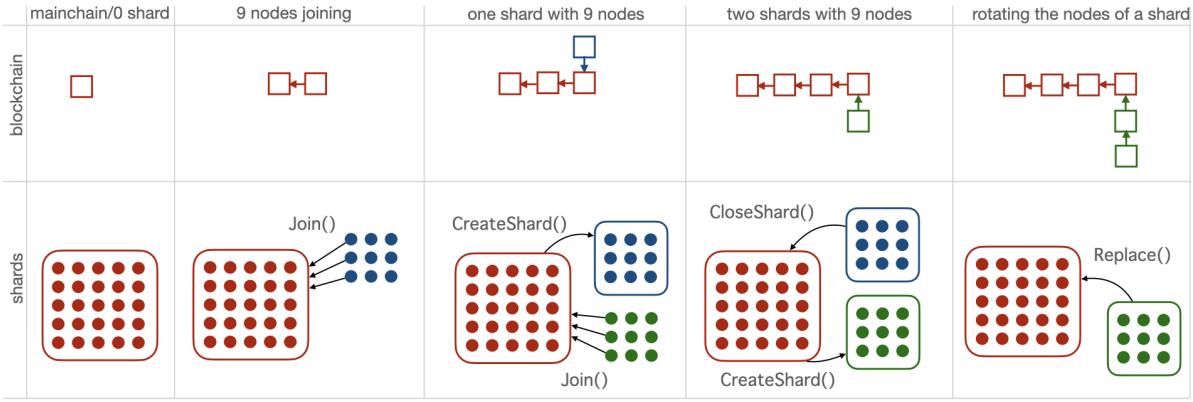

Figure 4.1: An example of the consecutive steps (from left to right) of a DApp-oriented dynamic transparent sharding execution where 2 shards are created and one of these shards is closed, and shard validators are rotated. Each shard executes DApps disjoint from other shards making our sharding protocol DApp-oriented and capable of fully executing DApps concurrently. As shard configurations are logged to the blockchain, it is transparent.

4.1.6 DApp-oriented Sharding

In the web-scale blockchain that we foresee, each DApp or a group of related DApps execute on at most one shard. This concept is known as DApp or service-oriented sharding [40]. For example, on a three-sharded system, we could have a Twitter DApp on the first shard, a Youtube DApp on the second shard, and an NFT DApp with multiple smart contracts (e.g., an ERC20 token and NFT smart contract) on the third shard. Validators in a shard deploy respective DApps on their shards. As a shard has N_v validators such that the Byzantine validators are less than a third with high probability (i.e., $f_{N_v} < N_v/3$), copies of the same DApp will not be deployed on multiple shards to disrupt the DApp-oriented nature of the sharding approach. Clients send requests to the shard that executes their required DApp. Each shard tag and the services they execute are made publicly available so the clients can connect to the shard they prefer to send their transactions. Due to the DApp-oriented nature of sharding, the state of each shard is disjoint, hence state consistency is not affected due to data migrations happening from the closing of multiple shards. Also, due to shard independence, there is no need for cross-shard transactions. We do not present our own cross-sharding protocol and it is out of the scope of the DApp-oriented dynamic transparent sharding protocol.

4.1.7 Transparency

Our sharding protocol works by invoking functions in smart contracts. As such, the CreateShard(·), Join(·), and CloseShard(·) functions invocations are registered in the blockchain as depicted in Figure 4.1. If the sharding protocol is implemented on a permissionless blockchain like SRBB as is the case in our evaluations, it allows anyone to query the blockchain and know the current and past sharding configurations.

4.1.8 Assumptions

We assume validators have static IP addresses and use these static IP addresses to identify other validators in the DApp-oriented dynamic transparent sharding protocol. This is for the sake of simplicity. Note that if required the implementation can be adjusted to use an Ethereum Node Record (ENR) [1].

Using IP addresses can be a Sybil risk if a single validator uses multiple IP addresses to join a shard by invoking the Join function. To prevent such issues, as mentioned previously, we adopt a stake-based strategy, which not only requires a validator to stake deposit prior to joining a shard but also requires a validator to register their wallet address in the sharding smart contract.

To prevent duplicate invocations to join a shard, the invoker's wallet address is stored in a map $called[senderAddr]$ where the key is the invoker's wallet address (i.e., $senderAddr$) and the value is a true or false Boolean value. The Join function can exit if $called[senderAddr]$ is set to true, preventing a Byzantine validator from joining the same shard with different IP addresses while using the same wallet address. A node is also prevented from invoking a shard function with multiple wallet addresses while parsing the same IP address. This is done by allowing only the registered wallet addresses on the sharding smart contract to invoke functions. Invocations from unidentified wallet addresses can be discarded.

The remainder of this chapter focuses on presenting our DApp-oriented dynamic transparent sharding protocol and its impacts on DApp performance. We defer additional security guarantees to future work.

4.1.9 Shard creation

The shard creation is presented in Alg. 4 and is deployed on the mainchain as a smart contract during the bootstrap of the blockchain. The *NumberOfAdmins* refers to the number of validators in the mainchain.

The shard creation smart contract is initialized with a set of data structures. The variable event refers to a broadcast message sent to all blockchain nodes in a shard. This event has a name (e.g., ShardNodes) and values that it broadcast (i.e. ShardNodes broadcasts an unsigned integer, a string array, and an address array). A *mapping* is a data structure mapping a key to a value. The *bool* is a boolean data type and $|\ admins\ |$ is the set containing the wallet addresses of mainchain validators.

[1] https://eips.ethereum.org/EIPS/eip-778

Algorithm 4 The smart contract that triggers the creation of a new shard

Initialization:
  event ShardNodes(uint, string[], address[])
  uint *shardSize*
  uint *NumAccounts*
  mapping (string → string[]) *shard*
  mapping (address → bool)) *called*
  mapping (string → bool)) *voted*
  mapping (uint → address[]) *accounts*
  mapping (uint → uint) *SizeOfShard*
  mapping (uint → uint) *NumberOfShards*
  mapping (uint → bool) *Created*
  admins : the set of addresses of admins
  NumberOfAdmins = | *admins* | ▷ *admins are the mainchain validators*
  mapping (uint → bool) *CloseInvoked*

SetSize(*val*, *NShards*): ▷ *threshold of admins/mainchain validators set shard size, number & accounts/shard*
  if *SenderAddr* ∈ *admins* **then** ▷ *if function invoker is an admin/mainchain validator*
    SizeOfShard[*val*]++
    NumberOfShards[*NShards*]++
    if *NumberOfShards*[*NShards*] == (*2* ∗ *NumberOfAdmins*/*3-1*) & *SizeOfShard*[*val*] == (*2* ∗ *NumberOfAdmins*/*3-1*) **then** ▷ *if shard size, number, accounts agreed*
      shardSize = *val*
      NumShards = *NShards*

Join(*ipAddr*): ▷ *when a node wants to join any shard as a validator*
  if *called*[*senderAddr*] == false & *voted*[*ipAddr*] == false **then** ▷ *prevents assigning IP twice to shard and making duplicate invokes from the same account/wallet*
    called[*senderAddr*] = true
    voted[*ipAddr*] = true
    random = *RANDContractaddr*.GetRand() ▷ *Fetch random number from RANDAO*
    shard[(*random*) mod (*NumShards*)] ∪ *ipAddr* ▷ *assigns the IP to a random key in the shard map*
    accounts[(*random*) mod (*NumShards*)] ∪ *senderAddr* ▷ *assigns the wallet address to a random key in the accounts map*
    if length(*shard*[(*random*) mod (*NumShards*)]) == *shardSize* & *Created*[*tag*] == false **then** ▷ *tag = (random) mod (NumShards)*
      CreateShard((*random*) mod (*NumShards*),*shard*[*random* mod (*NumShards*)],
        accounts[*random* mod (*NumShards*))
      length(*shard*[*random* mod (*NumShards*)]) = 0 ▷ *reset the shard tag value to 0*
      length(*accounts*[*random* mod (*NumShards*)]) = 0 ▷ *reset the shard tag value to 0*

ClosedShards(*tag*): ▷ *mainchain validator (admin) calls this if received n-t COMMITS for close from shardchain*
  if *SenderAddr* ∈ *admins* **then** ▷ *if the invoker is a mainchain validator/admin*
    CloseInvoked[*tag*] + +
    if *CloseInvoked*[*tag*] == (*2* ∗ *NumberOfAdmins*/*3-1*) **then**
      Closed[*tag*] = true

CreateShard(*tag*, []*shard*), []*accounts*):
  emit ShardNodes(*tag*, *shard*, *accounts*) ▷ *emit ip addresses & accounts of shard nodes, triggers shard start*
  Created[*tag*] = true ▷ *Assign shard as created*

Admins (i.e., validators of the mainchain) start by setting the size of shards and the number of shards (Alg. 4 line 16). Note that a threshold of admins should agree to the same settings for these values to be set (Alg. 4, line 20), and a threshold of admins can again agree to change these values during runtime making the sharding dynamic. Unlike other sharding schemes, we provide the capability to change shard size and number of shards even when the number of validators in the network remains constant (no validators joining or leaving).

Validators not in the mainchain invoke the Join function and parse their IP address (Alg. 4, line 24) in an attempt to join a shard. Note that, line 25 of Alg. 4, prevents validators from attempting to join multiple shards as well as two validators from joining shards with the same IP address. Subsequently, the shard creation contract fetches a random number *random* from a verifiable random number generation contract (Alg. 4, line 28) taken out of our system (We rely here on the RANDAO implementation [125] of a random number generator for the sake of explaining our protocol). Any other random number generation solution can also be used instead [126]).

Based on *random*, the IP address of a validator is assigned to a random key of the *shard* mapping (Alg. 4, line 29). Deriving the key values as (*random*) mod (*NumShards*) ensures that the IP address of a validator is assigned a shard tag x such that $x \in \{0, 1, ..., NumShards\}$. Similarly, the wallet address of the validator is also added to a random key corresponding to a shard tag of an *account* mapping.

We underscore that *NumShards* can be adjusted by admins to accommodate more, or fewer shards in the system. If for a particular shard key/tag the maximum number of validators (i.e., *ShardSize*) that could be assigned is complete, the shard creation contract invokes the function CreateShard, parsing an array of validators and validator accounts for a shard tag (Alg. 4, line 33).

The CreateShard function emits a smart contract event *ShardNodes* (i.e., a broadcast to all validators) with the validator's IP addresses and accounts that should be in a particular shard tag (Alg. 4, line 44).

Validators, upon receiving the same *ShardNodes* event from $f_{main} + 1$ validators such that $f_{main} < NumberOfAdmins/3$, verify its IP address is included in the smart contract event. If included, the validator candidates reconfigure and form a validator committee for a shard with a specific tag. Validators in a shard are known as shard validators. Details of this process are outlined in Alg. 5.

Algorithm 5 The algorithm executed by validators to create a new shard upon receiving event

```
1: Upon recv. same contract event from f_main + 1:
2:   if localIP ∈ event then                          ▷ If local IP is in event
3:     tag, shard, accounts ← extract(event)          ▷ extract values from event
4:     stop(node)
5:     editGenesis(accounts)                          ▷ Edit genesis with accounts
6:     connectPeers(shard)                            ▷ connect with other members of the shard
7:     start(node)
```

4.1.10 Shard closing

Shard closing is a procedure that helps prevent resource wastage. If a shard is not processing many transactions or is idle for a while, shard nodes can decide to close the shard. This is a part of the extended dynamism our protocol provides.

Algorithm 6 The smart contract that triggers the closing of a new shard

```
1:  Initialization::
2:      event Bs(string y)
3:      event COMMIT(string x, string m)
4:      mapping(string → uint) threshold
5:      uint Nv
6:      bool reached
7:      mapping(bytes32 → string) called;
8:      Nv = val                          ▷ The number of nodes in a shard from Algorithm 4
9:      ShardTag = tag                    ▷ tag of shard generated– based on RANDAO in Algorithm 4
                                          ▷ nodes call CloseShard parsing the state root
10: CloseShard(stateroot):
11:     if called[hash(SenderAddr, stateroot)] == true then   ▷ SenderAddr parsed stateroot before
12:         return                        ▷ avoids double voting
13:     called[hash(SenderAddr, stateroot)] = true            ▷ 'SenderAddr' parsed stateroot
14:     threshold[stateroot] = threshold[stateroot] + 1       ▷ number of nodes parsed specific 'stateroot'
15:     if threshold[stateroot] == 2*Nv/3+1 & !reached then   ▷ state root first reaching threshold
16:         reached = true
17:         emit COMMIT(stateroot, "COMMIT", ShardTag)        ▷ emits a commit event with the stateroot
18:     emit Bs(stateroot)                ▷ emits event with the parsed stateroot
```

Alg. 6 presents the smart contract algorithm for closing a blockchain shard. The variable N_v is the number of validators that the shard contains.

Firstly, once a shard validator decides to close the shard it is a part of, it invokes CloseShard (Alg. 6 line 10) parsing the state root the node prefers to close at. Alg. 6 lines 11- 12, ignores if a state root is parsed to the function by the same validator more than once. Otherwise, the *threshold* is increased (Alg. 6, line 14), which indicates the number of validators that have parsed a specific state root to the CloseShard function. At line 15 of Alg. 6, if $2 \cdot N_v/3+1$ nodes s.t. N_v is the total number of validators in the shard (i.e., shard validators) have parsed the same state root to CloseShard, then a COMMIT event is emitted with the *ShardTag*. Otherwise, the parsed state root to the CloseShard function is emitted in a smart contract event with the name Bs.

Alg. 7 presents the execution at a shard validator when either a COMMIT or a Bs smart contract event is received from the shard close smart contract algorithm (Alg. 6).

All shard validators subscribe to the close shard smart contract in their state. Upon receiving an event from the smart contract containing the address *CloseContractAddr*, the shard validator checks if the event is a COMMIT event (i.e., whether it contains the keyword COMMIT) at Alg. 7 line 3. If this condition is met, the current block number (Alg. 7 line 4) of the shard validator is retrieved and the state of the shard validator is traversed from the 0^{th} block to the current block to find the block number that contains the state root received in the event. If a

Algorithm 7 The algorithm executed by a participant to close a shard upon reception of the smart contract closure event

```
1:  Upon receiving a smart contract event:
2:    event ← subscribe(CloseContractAddr)          ▷ all nodes subscribe to closing smart contract
3:    if contains(event, COMMIT) then               ▷ smart contract event contain the "COMMIT" string
4:      number = getCurrentBlockNumber()
5:      for i = 0; i < number; i++ do
6:        block ← getBlock(i)
7:        if block.stateroot = event.stateroot then
8:          Close(block.number)                     ▷ parse closing block number to sync balances algo
9:          exit()                                  ▷ exit code
10:   else
11:     if nodeHas(event.stateroot) then            ▷ If node has same state root
12:       closeContractAddr.CloseShard(event.stateroot)   ▷ pass stateroot to SC
13:     else
14:       pending.push(event.stateroot)             ▷ push the stateroot to a pending array
15:       Check()

16: Check():                                        ▷ Do in parallel
17:   for i=0; i < length(pending); i++ do
18:     if nodehas(pending[i]) then
19:       CloseContractAddr.CloseShard(pending[i])  ▷ parse stateroot to smart contract
```

block exists with the received state root in the shard node, it decides to parse the block number to a Close function (Alg. 7 line 8) shown in Alg. 8 and exits.

If the *event* is not of type COMMIT but the shard validator has the state root contained in the event (Alg. 7 line 11), the validator invokes the CloseShard function in the Close shard smart contract parsing the state root. If the *event* is not of type COMMIT and the shard validator does not have the state root received, it is pushed to a pending array and kept (Alg. 7 line 14), in case the shard validator sees the state root sometime in the future. In Alg. 7 line 16, a *Check* function concurrently and repeatedly checks, if the shard validator has the pending state root. The CloseShard function is invoked parsing the state root if the state root is found (Alg. 7 line 19).

Algorithm 8 Shard chain participant Send Closing Account Balances to main chain

```
1:  Initialization:
2:    A is the set of account addresses in the shard
3:    Account(address, balance)                     ▷ A tuple of address and balance
4:    SA is the set of Account(address, balance) tuples

5:  Close(BNumber):                                 ▷ parse block number at which the shard should close
6:    for a ∈ A do
7:      b ← getBalance(a, BNumber)                  ▷ Balance of account a at closing block
8:      SA ∪ Account(a, b)
9:    Broadcast(SA, ShardTag)                       ▷ Broadcast to main chain nodes
10:   stop(shardNode)
```

Alg. 8 executes at each shard validator and retrieves balances of all accounts at the block number that the shard closes (Alg. 7 line 8) and broadcasts it and the shard tag to the mainchain validators (Alg. 8 line 9). Note that this broadcast is a reliable broadcast and waits for an ACK before the shard participants stop in the subsequent line.

Algorithm 9 Syncing of balance at the main chain from shard chains

1: **Initialization:**
2: $threshold = 2N/3 + 1$ s.t. N is the total number of shard chain nodes
3: mapping (bytes32 $\rightarrow$ uint) *count*

4: Receive(SA), $ShardTag$**:** ▷ *Receive Account tuple set*
5: $count[\mathsf{hash}(SA)] \leftarrow count[\mathsf{hash}(SA)] + 1$ ▷ *times specific account tuple set received*
6: **if** $count[\mathsf{hash}(SA)] == threshold$ **then** ▷ *If threshold of same SA received*
7: $CreateContractAddr$.ClosedShard($ShardTag$)
8: stop($node$)
9: editGenesis(SA) ▷ *Edit the genesis, adding accounts and balances tuple set*
10: start($node$)

A mainchain participant upon receiving the tuple set of accounts and balances SA from shard participants, and the shard tag, execute Alg. 9. Upon receiving SA, the algorithm keeps count of the number of unique SA sets received (Alg. 9 line 5). If $2 \cdot N_v/3 + 1$ number of the same SA set is received s.t. N_v is the number of shard validators in the closing shard, the mainchain node invokes the *ClosedShard* function in Alg. 4 to set the shard with the specific *tag* as closed. Consequently, the mainchain validators stop (Alg. 9 line 8), edit the genesis adding the new accounts and balances (Alg. 9 line 9) and restarts (Alg. 9 line 10). This way, the mainchain validators are synced with the accounts and balances of the shardchain. Note that, syncing accounts and balances from multiple shardchains upon shard closing does not affect the consistency of the state in the mainchain since states in each shard are disjoint as mentioned in Section 4.1.6.

4.1.11 Shard Committee Rotation

A shard with a particular tag remains active once it is created until it is closed. There is a risk of validators being bribed by a slowly-adaptive adversary while a shard is active. If sufficiently many validators in a shard committee are bribed this way (at least 1/3), there is a risk of shard takeover. To mitigate this risk, we propose a shard committee rotation protocol that is part of the shard creation smart contract (Alg. 4) but presented separately below for clarity. We consider an epoch as a specific time t where a shard committee processes transactions (during an epoch multiple blocks can be processed). At every t interval, all correct shard validators perform committee rotation correctly. Note that the number of correct validators in a shard is greater than $2N_v/3$.

The committee rotation starts with shard validators invoking Replace in the Alg. 1 Extension at line 4. Each correct shard validator parses the IP address of a shard validator they wish to replace within the shard and the shard tag simultaneously to the Replace function. For example,

Algorithm 1 Extension Shard committee rotation algorithm, a part of shard creation smart contract

1: **Initialization::**
2: mapping (string → uint) *ReplaceIpInvoked*
3: mapping (address → uint) *ReplaceAddressInvoked*

4: Replace($ipAddr, tag$): ▷ *Shard node parses its Ip address*
5: $ReplaceIpInvoked[ipAddr]$++
6: $ReplaceAddressInvoked[senderAddr]$++
7: **if** $ReplaceIpInvoked[ipAddr] > 2 \cdot N_v/3$ & $ReplaceAddressInvoked[senderAddr] > 2 \cdot N_v/3$ **then**
8: $called[senderAddr] =$ false
9: $voted[ipAddr] =$ false
10: $Created[tag] =$ false
11: $shard[tag] = shard[tag] \setminus ipAddr$ ▷ *remove IP from shard*
12: $accounts[tag] = accounts[tag] \setminus senderAddr$ ▷ *remove account/wallet address from shard*
13: Join($ipAddr$) ▷ *Invoke* Join *in Algorithm 4*

correct shard validators could decide to replace the shard validator that has been in the shard the longest. If a particular IP address and sender address have been used for the invocation $2 \cdot N_v/3$ times, the *called* and *voted* mappings are set to false for the corresponding IP address and sender address. Subsequently, the $Create[tag]$ is set to false and the Join function is invoked with the IP address. The Join function ensures shard validators are assigned to new shard committees following the same process of shard creation, that is, rotating the shards committees every epoch. Note that there is a checker that verifies the wallet address of the validator invoking the Replace function to prevent a Byzantine shard validator's invocation from being considered more than once during shard rotation in an epoch. However, these details are excluded in the algorithm for brevity.

At the end of an epoch, upon observing shard validators invoking Replace, the mainchain participants can adjust the number of shards and the number of validators per shard parameters according to the workload, which will change the number of shards and the validators per shard, making our sharding approach dynamic.

4.2 Availability

Shard validator rotation in every epoch is essential to tolerate bribery from a slowly-adaptive adversary. However, frequent changes of validators are a challenge when the state is sharded. New shard validators need to sync the state from previous shard validators, to service the DApps for a particular shard tag. This syncing process involves downloading the entire blockchain from previous nodes and is an expensive task, which is known to be a bottleneck on performance and affects the availability of shard nodes for transaction processing [84]. Since our sharding approach is Byzantine fault tolerant, downloading the latest state would suffice by querying $f+1$ previous shard committee members. There is no need to download the entire state history (i.e. snapshots) or the entire block history. As such, we provide better availability than some sharding approaches that shard the state [31].

4.3 Proof sketches

The first lemma shows that each shard contains less than $N_v/3$ Byzantine validators with high probability when the total number of validators in the network is n where $f < n/4$ are Byzantine. This is key to guaranteeing agreement among each shard to ensure that the view of the blockchain is consistent across all replicas. Note that while we prove by hand that each shard contains less than $N_v/3$ Byzantine nodes with high probability, concomitant works like OmniLedger [31] implicitly make this assumption. They assume that random shard assignment ensures the ratio between Byzantine and correct validators in any given shard closely matches the ratio of Byzantine and correct validators in the entire network with high probability. For simplicity in the analysis, we assume that the shard validators correspond to a sample of N_v validators taken uniformly at random among the whole set of n validators, and we reuse the same reasoning as in [71].

Lemma 3. *In each shard of N_v validators, there are less than $N_v/3$ Byzantine validators with high probability when the network has a total of n validators and f Byzantine validators s.t $f < n/4$.*

Proof. By assumption, we return a validator taken uniformly at random among all n validators. Consider each of these events as a Bernoulli trial such that a random variable X_i is 1 if the returned validator is correct and 0 if it is Byzantine. Let ρ be the portion of Byzantine validators. Because there are at most $f < n/4$ Byzantine validators among the initial n validators, we have $\rho < 1/4$.

$$\begin{aligned} \Pr[X_i = 1] = &\quad p = \tfrac{(n-\rho)}{n}, \\ \Pr[X_i = 0] = &\quad 1 - p = \tfrac{\rho}{n}. \end{aligned}$$

The random variable $X = \sum_{i=1}^{N_v} X_i$ thus follows a binomial distribution and $\Pr[X = k] = \binom{N_v}{k}\rho^{N_v-k}(1-\rho)^k$, hence we can derive the probability $\Pr[X \leq 2N_v/3]$ of creating a shard with less than 2/3 of correct validators:

$$\Pr[X \leq 2N_v/3] \quad = \quad \sum_{k=0}^{2N_v/3} \binom{N_v}{k} \rho^{N_v-k}(1-\rho)^k.$$

As this probability decreases exponentially fast with N_v there exists a parameter λ and a constant n_0 for which $\Pr[X \leq 2N_v/3] \leq 2^{-\lambda}$ for all $N_v \geq n_0$. As a result, each shard contains at most $\lceil N_v/3 \rceil - 1$ Byzantine validators with high probability, which concludes the proof. □

Given Lemma 3 and our threat model, we know that validators in a shard agree when less than $N_v/3$ are Byzantine. Hence each time a new block is added to a shard that did not fail, then the shard remains consistent with high probability. As a result, transparent access to sharding information remains guaranteed. An important remark is that the proof of Lemma 3 relies on having $N_v \geq n_0$, however, for the sake of the empirical analysis, we choose N_v relatively small (200 validators) in Section 4.4 to limit the cost of our AWS experiments.

Lemma 4. *If* $2 \cdot N_v/3 + 1$ *of the validators of shard* s *invoke its* CloseShard() *function with the same argument, then the shard* s *eventually closes.*

Proof. The state root at which a shard validator wishes to close the shard is received by all correct validators in the shard by the Bs event (Alg. 6, line 18) since the network is partially synchronous. Also, if a shard validator agrees to close the shard at a particular state root after seeing the state root event, they will either have that state root in their history or will eventually have it since consensus ensures the nodes have the same state history eventually. Therefore, if at some point in time, $2 \cdot N_v/3 + 1$ validators (Alg. 6 line 15)—where the number of Byzantine validators is $f < N_v/3$, agree on the state root, a commit event will be emitted (line 17) triggering the close of the shard. □

4.4 Evaluation

In this section, we evaluate the performance of our DApp-oriented dynamic transparent sharding protocol. Our sharding approach was implemented on SRBB [1]. Note that while we evaluated our sharding protocol on SRBB to benefit from the fork-free guarantees, our solution is adaptable to any Ethereum-based blockchain executing a BFT consensus. We implemented our smart contract algorithms using Solidity. All the experiments were performed on AWS with c5.2xlarge (8 vCPUs, 16 GB RAM) EC2 instances which have similar performance to a modern PC. The AWS instances were evenly distributed across 10 AWS regions which included 5 continents. Namely, we used the Bahrain, Cape Town, Milan, Mumbai, Sao Paulo, Ohio, Oregon, Stockholm, Sydney, and Tokyo AWS regions. All evaluations were performed using the DIABLO blockchain benchmarking suite [28]. We used the NASDAQ, Uber, and FIFA DApp workloads to present the capability of our DApp-oriented dynamic transparent sharding protocol to concurrently execute different DApps (Sections 4.4.1).

4.4.1 Concurrently Executing DApps in Different Shards

Figure 4.2 presents the average throughput at load of 200 SRBB validators spread across 10 AWS regions against sharded SRBB consisting of 3 shards with 200 SRBB validators per shard. When an aggregation of the NASDAQ, Uber, and FIFA workloads are sent to SRBB, we observe that SRBB only achieves an average throughput of 1451.18 TPS. SRBB also loses 9% of the transactions sent. This is expected, due to the congestion caused by 3 demanding DApp workloads.

With 3 shards of SRBB each executing the NASDAQ, Uber and FIFA DApp workloads [28] concurrently, the sharded version of SRBB achieves an average throughput of 2828.87 TPS. The higher average throughput in comparison to SRBB is expected since each SRBB shard processes a unique set of transactions, without performing cross-shard transactions because, as mentioned previously, we follow a DApp-oriented sharding approach. In contrast to SRBB, sharded SRBB loses only 2% of the transactions sent.

Figure 4.3 presents the peak throughput when (1) SRBB executes the aggregated workload consisting of NASDAQ, Uber, and FIFA workloads and (2) sharded SRBB receive the NASDAQ,

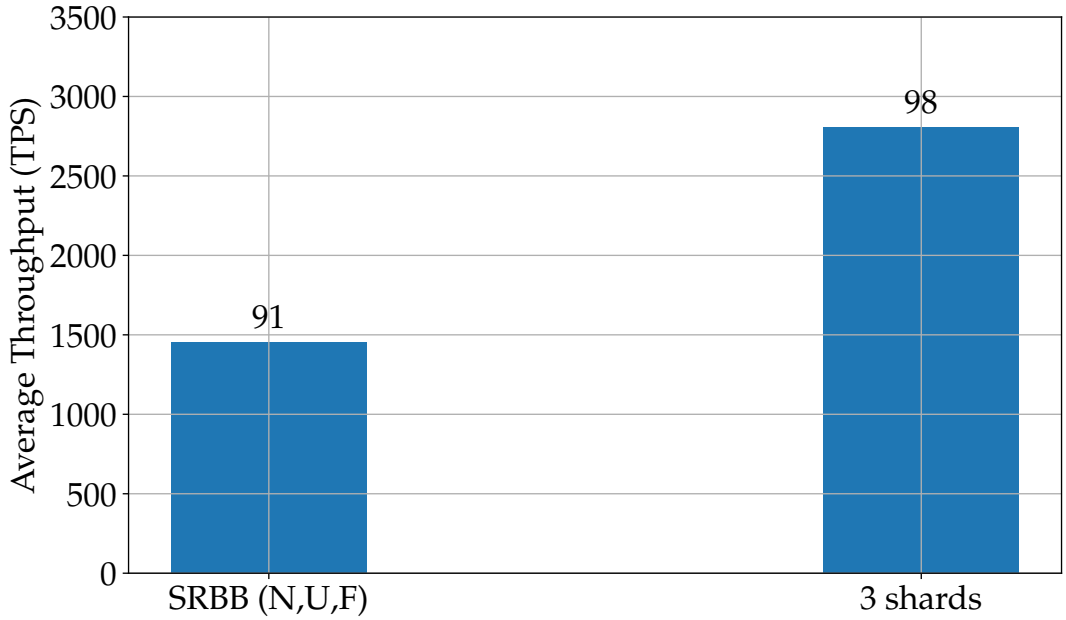

Figure 4.2: The average throughput of 200 SRBB validators executing the aggregation of NASDAQ (N), Uber (U) and FIFA (F) workloads compared with sharded SRBB consisting of 3 shards (each shard has 200 validators). The first shard executed the NASDAQ workload, the second shard executed the Uber workload and the third shard executed the FIFA workload. The execution of the 3 workloads in sharded SRBB was concurrent. When SRBB executed the aggregated workload, only 91% of transactions were committed. It also performed significantly lower than sharded SRBB.

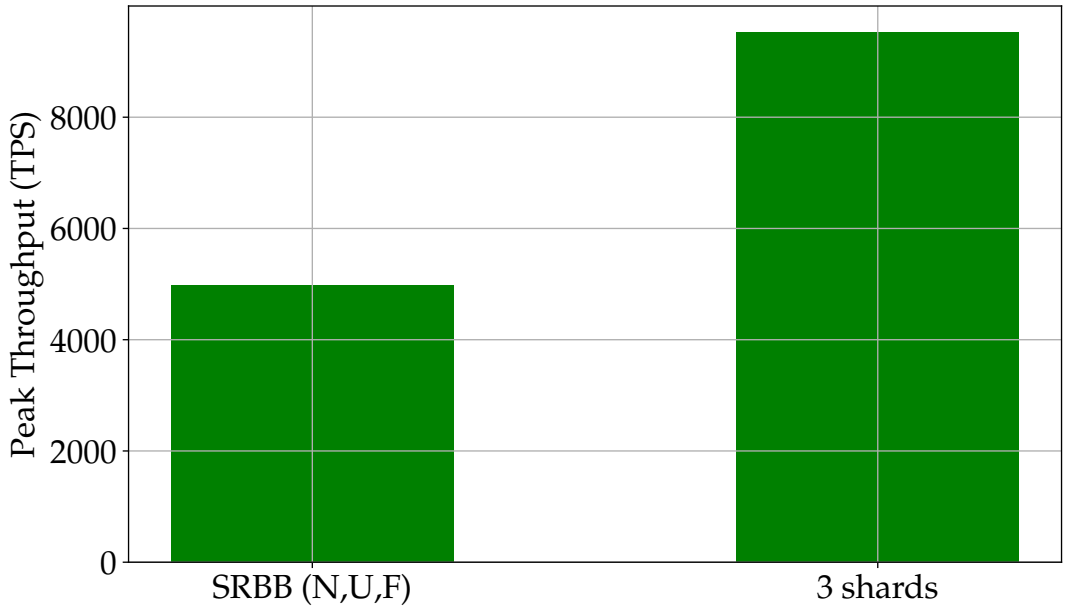

Figure 4.3: The peak throughput of 200 SRBB validators executing the aggregation of NASDAQ (N), Uber (U), and FIFA (F) workloads compared with sharded SRBB consisting of 3 shards (each shard has 200 validators). In sharded SRBB, the first shard executed the NASDAQ workload, the second shard executed the Uber workload and the third shard executed the FIFA workload. The execution of the 3 workloads in sharded SRBB was concurrent. A peak throughput of 4986 TPS was achieved when SRBB executed the aggregated workload. In contrast, sharded SRBB achieved a peak throughput of 9523 TPS.

Uber and FIFA workloads on shard 1, 2 and 3 respectively. SRBB only achieves a peak throughput of 4986 TPS. In contrast, sharded SRBB reaches a peak throughput of 9523 TPS, which is almost twice as much as SRBB. Executing DApps in separate shards allow sharded SRBB to reduce congestion, and improve blockchain performance compared to SRBB.

4.5 Summary

In this chapter, we introduced DApp-oriented dynamic transparent blockchain sharding, a DApp-oriented sharding protocol that executes DApps concurrently and has the ability to change its sharding configuration at runtime without hard forks. Our implementation relies on smart contracts, hence anyone can double-check the effectiveness of the sharding configuration by auditing the current state of the blockchain.

Most importantly, the performance of our worldwide geo-distributed evaluations demonstrates that the DApp-oriented dynamic transparent sharding improves throughput under a balanced workload (Figure 4.2). With just 3 shards where each shard is dedicated to executing a single DApp: NASDAQ, Uber and FIFA respectively, our sharding protocol on SRBB (sharded SRBB) can achieve a total average throughput of 2828.87 TPS and a peak throughput of 9523 TPS. In contrast, executing SRBB alone halved the average throughput to 1451.18 TPS and the peak throughput to 4986 TPS compared to its sharded counterpart. This shows that sharded SRBB built upon the DApp-oriented dynamic transparent sharding protocol can support multiple realistic DApps by concurrently executing them on separate shards.

We have thus far improved blockchain performance for DApp execution by presenting SRBB and a sharding protocol. SRBB outperformed 6 modern blockchains under realistic DApp workloads. The sharded version of SRBB which used our sharding protocol, yielded better performance over SRBB and supported the concurrent execution of multiple DApps. In addition to blockchain sharding, other avenues have shown promise in enhancing blockchain performance such as Layer 2 solutions [29] and the decoupling of blockchain consensus and state executions [32, 33]. As this thesis focuses on improvements in the blockchain layer, in the next chapter, we focus on another method to enhance the DApp performance of SRBB which involves decoupling SRBB.

Chapter 5

Collachain: Decoupling Smart Redbelly Blockchain

In this chapter, we present a more performant version of SRBB by decoupling its SRBB VM and consensus into two separate machines to improve the performance of DApps under burst workloads compared to SRBB. Support for burst workloads is important to widen the adoption of Web3 as stock markets like NASDAQ have sporadic bursts in requests.

Decoupling has been used in various forms in blockchains for varying purposes [32, 33]. For example, PRISM [32] decouples blocks into separate chains to improve the performance of the longest chain rule used in Nakamoto's consensus. Hyperledger Fabric [33] decouples transaction ordering and execution using separate order nodes and peer nodes. Decoupling transaction ordering facilitates different ordering services (i.e., consensus protocols) to be plugged into the system. Similar to Fabric, we decouple our consensus and execution to produce Collachain. However, unlike Fabric, the main motivation behind our decoupling is to enhance blockchain performance rather than to offer flexibility in using a customized consensus protocol. Fabric and Collachain are inherently different in their architecture. More specifically (1) Fabric follows a UTXO model whereas Collachain follows a balance model and (2) Fabric's client nodes submit transactions for execution and for consensus in two separate rounds whereas Collachain simply submits transactions to the SRBB VM.

In summary, this chapter presents the following contributions:

- We present the decoupled version of SRBB known as Collachain which separates the SRBB VM and the consensus of SRBB into two separate machines.
- We evaluate Collachain against SRBB using the DIABLO blockchain benchmark suite [28]. Collachain shows a 33% increase in peak throughput compared to SRBB for the NASDAQ burst workload [28].
- As we already evaluated real DApp workloads throughout this thesis, we show that Collachain is well-suited for burst native payment workloads (a timely requirement for the wider adoption of Web3) and scales up to 200 nodes spread across 10 AWS regions spanning 5 continents. Collachain achieves an average throughput of 2038 TPS at 200 nodes

for burst native payment workloads. When Transport Layer Security (TLS) is enabled between nodes, Collachain achieves an average throughput of 1960 TPS, showing only a drop in 4% compared to its non-TLS counterpart.

Chapter Outline: The rest of this chapter is ordered as follows: Section 5.1 presents the model for Collachain, Section 5.2 presents Collachain, Section 5.3 presents the evaluation of Collachain, and Section 5.6 presents the discussion mentioning potential drawbacks. Finally, in Section 5.7, we conclude this chapter.

5.1 Model

System Model: For Collachain, we introduce the notion of a participant that is a pair of SRBB VM and consensus nodes. More specifically, a participant owns a SRBB VM and a consensus node (More specifically, 1 participant has 1 VM + consensus. There are n such participants). Therefore, when we say there are m nodes in Collachain, we mean there are $n = m/2$ participants each owning a node pair consisting of a consensus and SRBB VM node.

Collachain is a permissionless blockchain where (1) any node can join or leave the network and (2) any node has a chance to become a validator based on a periodic election process if they stake a certain amount of tokens. This membership approach of periodically rotating validators prevents an exclusive set of nodes from always being validators, providing a permissionless environment similar to Algorand [10]. A Collachain node can be (1) a client that sends transactions and reads the state of the blockchain (2) a validator that participates in consensus, executes transactions, and keeps a full state of the ledger to service clients, or (3) a candidate validator that is an applicant to the validator position.

Initially, Collachain is bootstrapped with an initial set of participants pre-specified in the genesis block through a KYC process. Potential future participants must first express interest by (1) depositing tokens (e.g. a pre-defined sum of ether) in a reconfiguration smart contract or (2) authenticating using a KYC process. Depositing tokens provides mitigation against Sybil attacks. Unlike SRBB, Collachain periodically elects participants by executing an election. Participants' consensus node and SRBB VM pair are voted upon in this election. The current set of participants elects the next set of participants using a reconfiguration smart contract. The details of the participants of a committee are registered in the smart contract after an election outcome and each Collachain validator gets to know of other validators in the committee through an event emitted by the reconfiguration smart contract.

Threat model: We assume that there are n Collachain participants such that at most f is *Byzantine* where $f < n/3$. Byzantine participants can act arbitrarily and deviate from the protocol. The periodic rotation of Collachain validators mitigates the validator committee from being bribed by a slowly-adaptive adversary.

Network Model: Our network consists of a set of n participants and $2n$ nodes. These $2n$ nodes are well-connected. We assume *partially synchronous* communication similar to prior

work, as consensus cannot be solved with asynchrony in the presence of Byzantine validators [47](Chapter 2).

5.2 Collachain

Collachain is a collaborative permissionless blockchain compatible with the largest ecosystem of DApps and is optimally resilient against Byzantine failures. The layered architecture of Collachain is depicted in Figure 5.1 with a SRBB VM node at the top, and a consensus node at the bottom. The communication between the consensus node and the SRBB VM node is event-based and implemented with gRPC. Although the event-based communication adds some communication overhead, since both the consensus node and the SRBB VM of Collachain can execute on separate machines, this offers better modularity and better performance compared to SRBB (Section 5.3).

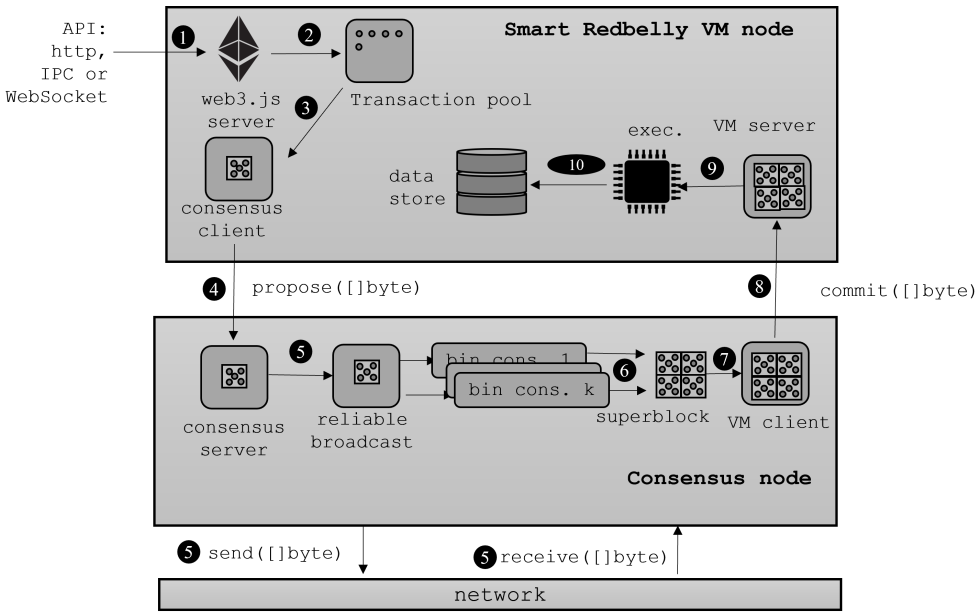

Figure 5.1: The architecture of Collachain. ❶ A client sends a transaction to some Smart Redbelly Blockchain VM node (SRBB VM), ❷ at each replica, the web3.js server eagerly validates transactions and sends them to the transaction pool that ❸ sends a block to the consensus client. ❹ The consensus client proposes it to the consensus protocol. Upon reception of a new block from the consensus client, ❺ the consensus server in the consensus node propagates it through the network with a reliable broadcast. Remote consensus nodes start participating in the same instance (if not done yet) upon reliably delivering this proposed block. ❻ When the consensus outputs some acceptable blocks, all of these blocks are combined into a superblock ❼. The VM client sends this superblock to the SRBB VM by invoking commit ❽. The VM server upon receiving the superblock sends the block to be executed ❾. After execution, the block is appended to the ledger and stored in the data store ❿.

5.2.1 The Transaction Lifecyle

Since Collachain is the decoupled version of SRBB, there are slight variations between the architecture and the transaction lifecycles of these two blockchains. Since the SRBB VM and consensus are decoupled in Collachain across two machines, RPC invocations occur between the consensus instance and the state machine instance for communication. To facilitate RPC calls, additional components were added to Collachain's system architecture that was not present in SRBB.

The lifecycle of a transaction in Collachain goes through the stages below:

1. **Reception.** The client creates a properly signed transaction and sends it to at least one SRBB VM node. Once a request containing the signed transaction is received ❶ by the JSON RPC server of SRBB VM, the eager validation (Chapter 2) starts. If the validation fails, the transaction is discarded. If the validation succeeds, the transaction is added to the transaction pool ❷. Unlike modern blockchains where the transaction is propagated to all validators increasing the number of eager validations, Collachain includes transactions in blocks directly from the transaction pool. This is because Collachain includes TVPR from SRBB. Once a threshold of transactions has been received, the SRBB VM serializes blocks created from the transaction pool and sends them to the consensus client ❸.

2. **Consensus.** Once the consensus client receives a proposed block, it sends the corresponding byte array to the consensus system by invoking the `propose([]byte)` method using gRPC ❹. The consensus server upon receiving this byte array starts a new instance of consensus using the new block if it is not currently part of another consensus instance and reliably broadcasts the block to the network of consensus nodes ❺. Otherwise, it adds the new block to the block queue waiting for the current consensus instance to terminate. The consensus execution then invokes a *binary* consensus instance for each reliably delivered block. The output of the binary consensus instance indicates the indices of acceptable blocks ❻ as detailed before in Alg. 2, line 18. The consensus system creates a superblock with all acceptable blocks (Alg. 2, line 19) and sends it to the VM client ❼. The VM client sends the superblock to the SRBB VM via gRPC by invoking the `commit([]byte)` method ❽.

3. **Commit.** When the superblock is received by the VM server on SRBB VM, the superblock is first deserialized using JSON unmarshalling. Subsequently, the superblock is passed on to the execution module of the SRBB VM ❾. The execution of transactions in Collachain is identical to SRBB. The SRBB VM lazily validates and executes transactions by iterating through every block in the superblock and every transaction in each block (Chapter 3, Section 3.2.4). Finally, the SRBB VM appends each block to the ledger in the datastore ❿.

5.3 Evaluation

In this section, we present the evaluation of Collachain. First, we present a comparison of performance between Collachain and SRBB. Next, we evaluate the scalability of Collachain for the DIABLO NASDAQ workload.

In summary, we show that Collachain yields an improvement of 33% in peak throughput over SRBB. We also show that Collachain can commit 100% of the NASDAQ workload transactions within 12 seconds whereas SRBB takes 15 seconds to commit 100% of the transactions. Finally, we show that Collachain scales up to 200 machines (i.e., 100 participants) achieving an average throughput of $\sim$ 2000 TPS.

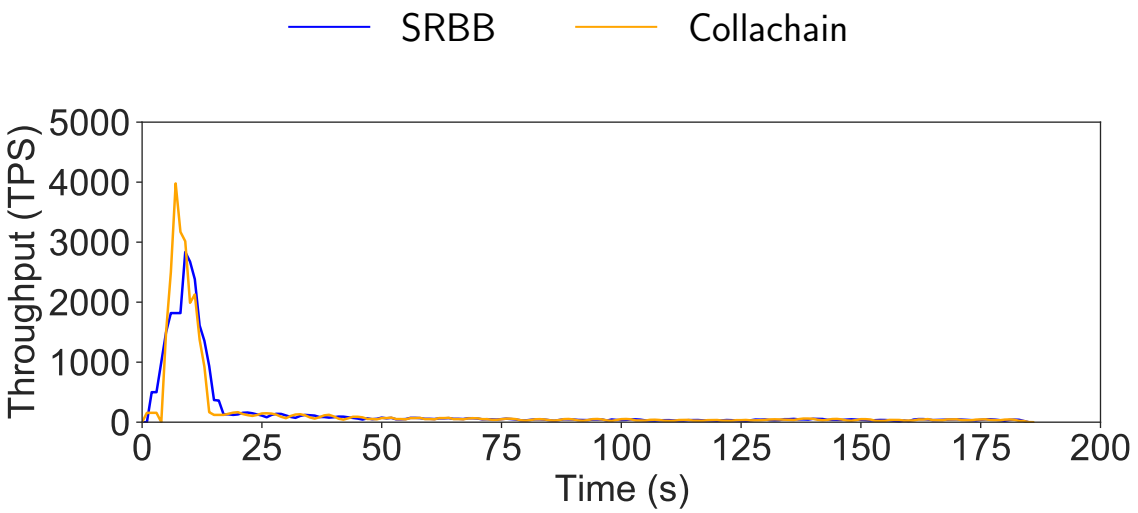

Figure 5.2: The throughput over time of Collachain (The decoupled version of SRBB) and SRBB for the NASDAQ workload on 200 machines.

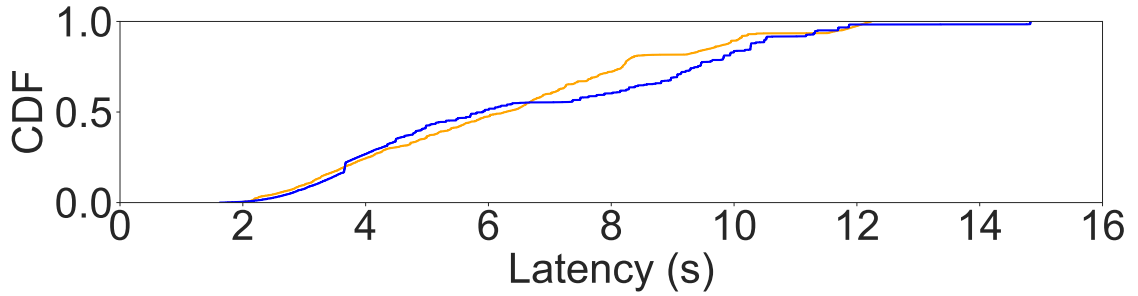

Figure 5.3: The CDF latencies of Collachain and SRBB for the NASDAQ workload on 200 machines.

5.4 Collachain vs SRBB

Below we present a comparison of performances between Collachain and SRBB. We deployed 200 c5.2xlarge (8 vCPUs, 16 GiB of memory) geo-distributed machines spanning 10 AWS regions on 5 continents. Namely: Bahrain, Cape Town, Milan, Mumbai, Sao Paulo, Ohio, Oregon, Stockholm, Sydney, and Tokyo. The 200 machines of Collachain consisted of 100 consensus nodes and 100 SRBB VM nodes, thus, the evaluation of Collachain consisted of 100 participants each owning a consensus and SRBB VM node. We evenly distributed the machines such that 10 consensus nodes and 10 SRBB VM nodes existed per AWS region and each consensus and SRBB VM node pair that worked together as a participant existed in the same AWS region. In contrast, for SRBB, there were 200 SRBB nodes across 10 regions where 20 SRBB nodes were located in a single region. The total number of machines used for both blockchains was kept equal to ensure the computational power used in both cases is equal and the comparison is fair as can be. We elaborate more on the fairness of this evaluation as a part of our discussion (Section 5.6).

Figure 5.2 shows the throughput over time for Collachain and SRBB for the NASDAQ workload. To produce a smooth curve (i.e., instead of a saw-toothed curve), the throughput is averaged over 3 seconds, and depicted as a function of time every 3 seconds. Collachain reaches a peak throughput of 4000 TPS whereas SRBB reaches a peak throughput of ∼3000 TPS. Thus, Collachain improves the throughput of SRBB by 33% for the same number of machines. Figure 5.3 shows the Cumulative Distributed Function (CDF) latencies of both Collachain and SRBB. We observe that Collachain commits 100% of the transactions within 12 seconds whereas SRBB takes 15 seconds to commit 100% of the transactions.

Executing consensus and SRBB VM on the same machine results in high CPU and memory under demanding transaction workloads. In the implementation, as separate Go routines handle execution and consensus, the true potential of concurrency is not achieved due to resource constraints. Decoupling helps to split the consensus and execution tasks into two separate machines, allowing the use of more CPU and memory for the same task, and speeding up performance.

5.5 Scalability of Collachain

To evaluate scalability, we deployed Collachain in 10 AWS regions spanning 5 continents. Namely: Canada, London, Mumbai, Oregon, Paris, São Paulo, Singapore, Stockholm, Sydney, and Tokyo. The SRBB VM nodes were of type c5.2xlarge with 8 vCPUs, 16 GiB of memory, and the consensus nodes were of type c5.4xlarge with 16 vCPUs, 32 GiB of memory. As we evaluated varying workloads throughout this thesis, we evaluated Collachain with a burst workload. This was to observe Collachain's capability to handle such workloads that we envision would be increasingly observable with the wider adoption of Web3. Each SRBB VM node received a burst workload of 1500 native payment transactions concurrently at a rate of 1500 TPS. As previously mentioned, each participant was considered to execute both a consensus node and a SRBB VM node. To show the impact of encryption on Collachain, we evaluate Collachain without end-to-end

encryption (w/o TLS) and with encryption (with TLS).

Figure 5.4 depicts the throughput of Collachain with an increasing number of nodes: We start our experiment with 20 machines spread evenly in 10 AWS regions and add machines by groups of 20 evenly spread across the same 10 AWS regions until we reach 200 machines (i.e., 100 participants). We observe that the throughput increases as we increase the number of nodes from 1100 TPS at 20 machines to 2038 TPS at 200 machines, demonstrating the scalability of Collachain in a geo-distributed setting. The curve flattens out at a large scale between 140 and 200 nodes, indicating that the gain obtained in throughput is less effective when adding more machines. This is due to an increase in the number of machines consuming the available bandwidth. Finally, we observe, as expected, that TLS encryption comes at a cost. However, this overhead is negligible in comparison to the overall performance as the peak throughput with TLS (1960 TPS) is only 4% lower than the peak throughput without TLS (2038 TPS).

Figure 5.5 shows the latency of transactions of Collachain in the aforementioned geo-distributed environment as the number of nodes increase. We observe that the latency increases with the number of nodes. We observe similar minimum latencies across all system sizes but the 99^{th} percentile indicates that some requests can take much longer, especially at large scale: the transactions take less than 10 seconds to execute on up to 40 nodes while they take less than 40 seconds to execute at 200 nodes. It is important to note that these latencies can be viewed as the time for a transaction to become final: thanks to our deterministic byzantine fault tolerance consensus (Chapter 3), transactions are committed (and thus final) as soon as the consensus ends and the superblock is executed and appended to the chain. This differs from classic blockchains [16, 13] whose consensus is reached after the block is appended and after more "block confirmations" occur. Interestingly, this increase in latency as the number of machines increases does not prevent the throughput of Collachain from scaling as seen by Figure 5.4. This is due to the superblock optimization: As more machines participate, more blocks get proposed, and running consensus takes more time, which increases the latency, however, the number of transactions decided per consensus instance also increases, which provides scalability.

5.6 Discussion

In this section, we discuss the potential drawbacks of decoupling SRBB to produce Collachain. We also discuss the fairness of the comparison between SRBB against Collachain.

5.6.1 Fault Tolerance of Collachain

For Collachain to achieve the same fault tolerance as SRBB, the consensus node and the SRBB VM node should be considered as belonging to the same participant (i.e., entity) such that $f < n/3$ participants are Byzantine where the total number of participants is n. A limitation with decoupling is if we consider the SRBB VM nodes and consensus nodes separately then the fault tolerance will be much lower than SRBB. This is because the consensus nodes alone as a separate entity can only tolerate a third of its nodes failing and with the addition of failures

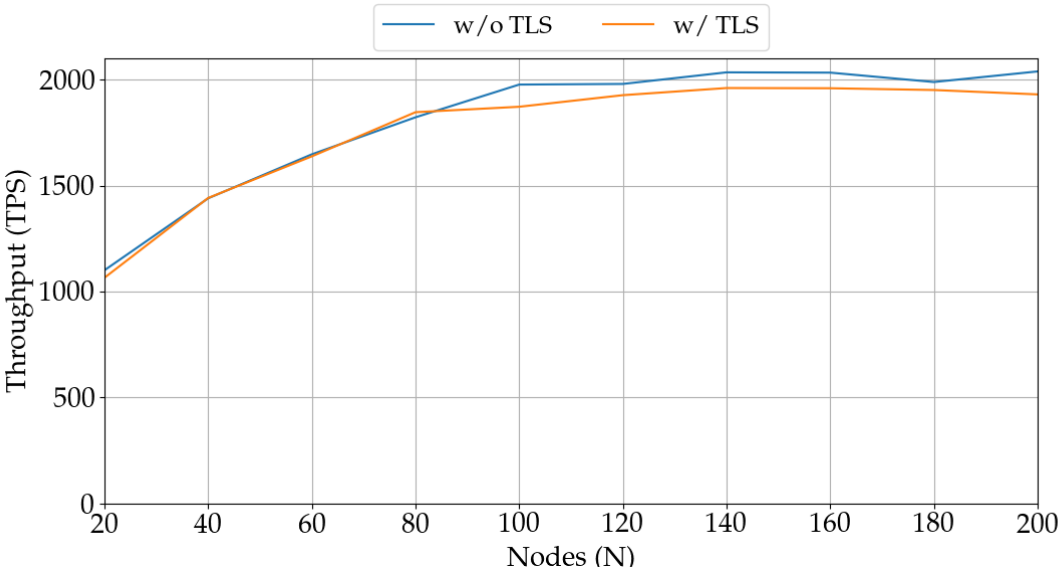

Figure 5.4: The avg. Throughput of Collachain with and without TLS

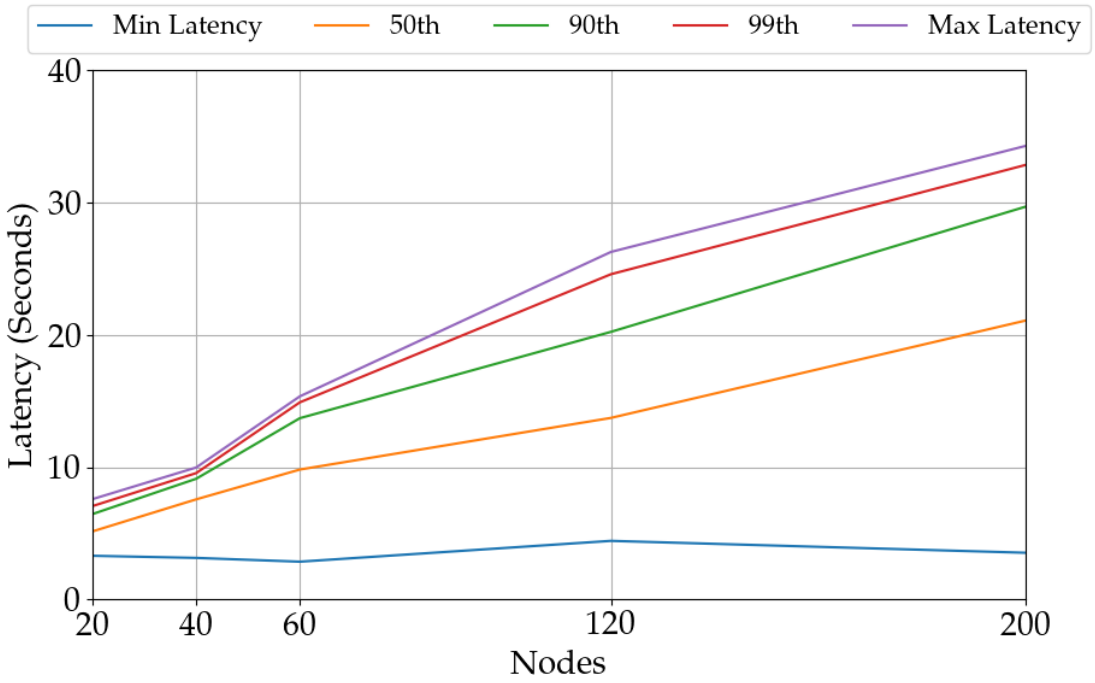

Figure 5.5: The percentile Latencies of Collachain

from SRBB VM nodes, Collachain will have a lower fault tolerance (i.e., inability to recover from a low number of failures).

5.6.2 Additional Resource Usage

Decoupling can incur additional resource usage. With SRBB, a single user can use a single machine to join as a SRBB validator whereas with Collachain a single user requires two machines to execute a consensus and a SRBB VM node separately.

5.6.3 SRBB vs Collachain Comparison Fairness

Although in our evaluation we compare SRBB's performance to Collachain, one may argue such a comparison is not fair. We used the same number of total machines when comparing the performance of SRBB and Collachain to ensure that decoupling itself helps performance rather than an increase in the number of machines that increases the processing power. However, one might argue that to compare Collachain to SRBB fairly, the consensus nodes in the Collachain should equal the nodes of SRBB if consensus is the bottleneck. However, we see that Collachain's consensus scales as the number of consensus nodes and SRBB VM nodes increase (Figure 5.4). Thus, showing that consensus is not a bottleneck. It is true that comparing a blockchain to its decoupled counterpart can be unreasonable. As the saying goes, we want to compare apples with apples. However, our comparison is merely to distinguish that decoupling on SRBB is effective and is a method that has the potential to improve performance with the same amount of available resources (i.e., machines).

5.6.4 Choice of DApp Workload

To compare Collachain with SRBB we used the NASDAQ workload. One may incorrectly think that we cherry-picked NASDAQ to show the performance enhancements of Collachain over SRBB (Figure 5.2 and Figure 5.3). As our focus was to evaluate the gain in performance for DApps under burst workloads, we believe our evaluation was fair as the NASDAQ workload is a realistic DApp burst workload with a peak transaction sending rate of ∼20K TPS [28]. We intend to evaluate Collachain with a wide variety of realistic DApp workloads as a part of our future work.

5.7 Summary

In this chapter, we presented Collachain – a decoupled version of SRBB. Our experiments demonstrated that Collachain can increase the peak throughput of SRBB by 33% and can decrease SRBB's latency by 3 seconds for a real DApp workload. We further demonstrate that Collachain scales well up to 200 machines spanning 5 continents and 10 AWS regions.

In this thesis, we have thus far focused on enhancing blockchain performance for DApp executions (Objective 1) to widen the adoption of Web3. While blockchain performance is important, blockchain security must also be considered for the wider adoption of Web3. DApps

offer decentralization to mitigate the centralization drawbacks such as the single point of failure and data censorship brought about by centralized web applications [25]. However, the execution of DApps can be centralized if a governance oligarchy in the blockchain dictates DApp executions. This can lead to insecure executions (e.g., double spending attacks) or censorship defeating the benefits that DApps and Web3 offer. Therefore, in the next chapter, we focus on enhancing blockchain security for DApp executions (Objective 2) by presenting a blockchain governance protocol to mitigate an oligarchy.

Chapter 6

Blockchain Proportional Governance: Mitigating a Governance Oligarchy

In this chapter, we focus on enhancing blockchain security for DApp executions to widen the adoption of Web3. An oligarchy in blockchain governance can dictate DApp executions or lead to dissident executions of DApps resulting in either censorship or double spending attacks. We enhance blockchain security by mitigating such an oligarchy in blockchain governance.

The notion of *governance*, which is generally understood as the processes relied upon to make decisions and modify the protocol, has become an important topic in blockchain [127, 128, 54]. In the context of blockchains, governance can include decisions such as updating the blockchain protocol, varying blockchain parameters (e.g., changing the block period), and deciding upon a block to be executed (e.g., solving consensus) [54, 129]. The absence of governance has led users to create dissident instances of the two largest blockchains: Bitcoin is now split into BTC and BCH while Ethereum is now split into ETH and ETC [130, 131].

A pernicious threat in blockchain governance is the risk of an attacker controlling an *oligarchy* amongst the governors. If this happens the oligarchy can dictate the decisions to modify the blockchain protocol making the blockchain governance centralized, and the DApp executions in the blockchain centralized.

An oligarchy can be formed in blockchain governance through the governance election process. More specifically, modern blockchains, which have mostly replaced Proof-of-Work (PoW) with Proof-of-Stake (PoS), elect governors based on their stake providing more opportunities to those that have a higher stake to elect governors [10, 82, 54, 64]. Given the skewed distribution of wealth, this can inadvertently create an oligarchy [63].

To mitigate the aforementioned method of forming an oligarchy of governors, we propose *proportional governance* that is compatible with smart contract supported (i.e., DApp-supported) blockchains.

The *proportional governance* tackles the formation of an oligarchy among governors through

the election process. Proportional governance elects *governors* that proportionally represent the voters. This is to prevent an adversary from creating an oligarchy in the governance. Proportionality is a concept widely used in social choice theory to elect a set of candidates fairly to a legislative body [132]. In general terms, proportionality ensures that a diverse set of candidates are elected ensuring even the minority voters are represented in a legislative body.

As multiple governors need to be elected to a blockchain governance committee, we needed a multi-winner election protocol. Thus, we used the Single Transferable Vote (STV) protocol [78], used for example to elect the Australian senate [133]. STV outputs a set of candidates proportionally representative of the voted preferences. However, the STV protocol is synchronous: a voter simply has to cast a vote within a limited known period of time for its vote to be counted when tallying votes. Blockchains instead operate in a general network (e.g., the Internet) where the communication is not synchronous and where Byzantine nodes can arbitrarily delay messages. Thus, the STV protocol executing on a blockchain with n nodes that waits for votes from all n nodes cannot progress if Byzantine voters do not cast votes. This is because one cannot distinguish a slow voter from a Byzantine voter due to the upper bound on the message delay being unpredictable.

To solve this problem, we develop a variant of STV known as BFT-STV that offers (1) the same proportionality guarantees as STV, (2) does not assume synchrony, and (3) works in a Byzantine setting s.t. at most $t < n/3$ of n voters are Byzantine (we denote $f \leq t$ as the actual number of Byzantine voters). The ratio of f comes from (i) the need for voters to reach consensus on the new set of governors and (ii) the impossibility of solving consensus with $f \geq n/3$ Byzantine participants in blockchains [134]. We implement BFT-STV in a smart contract to make our proportional governance pluggable and compatible with smart contract supported blockchains.

To the best of our knowledge (Table 6.1), the *proportional governance* we propose is the first solution that solves the proportional governance problem (Def. 2). Our solution (1) mitigates an oligarchy among governors and (2) is pluggable and compatible with smart contract supported blockchains due to its generality and smart contract based implementation.

In summary, this chapter defines the proportional governance problem (Section 6.2), designs a solution for it known as *proportional governance* that is compatible and pluggable with smart contract supported blockchains (Sections 6.3), proves the solution correct (Section 6.3.4) and evaluates the proportional governance solution (Section 6.4). Lastly, to complement the proportional governance that elects a set of governors, we present a governance reconfiguration protocol. Our proposed solution offers the following contributions:

- We introduce the first Byzantine fault tolerant multi-winner election protocol, called *BFT-STV* to elect a set of governors proportionally to solve the proportional governance problem (Def. 2) and to mitigate an oligarchy among governors (Section 6.3). BFT-STV is a new primitive that augments the STV election protocol to work in a setting where at most $t < n/3$ Byzantine voters exist among n voters without assuming synchrony (we denote $f \leq t$ as the actual number of Byzantine voters). As it is impossible to distinguish a non-

Blockchain	Election	Proportional Governance
Tendermint [135]	None	no
Algorand [10]	Sortition	no
Hybrid consensus [136]	PoW puzzle	no
Zilliqa [57]	PoW puzzle	no
OmniLedger [31]	Sortition	no
RapidChain [30]	PoW puzzle	no
ComChain [137]	None	no
Libra [138]	None	no
SmartChain [139]	None	no
Polkadot [54]	Multi-winner approval voting	no
EOS [64]	Multi-winner approval voting	no
This work (compatible with smart contract supported blockchains)	Multi-winner preferential voting	yes

Table 6.1: Blockchains do not solve the proportional governance problem (Def. 2)

responsive Byzantine voter from a delayed message, we introduce a new election quota $q_B = \frac{n-t}{k+1}$ where k is the size of the committee. Interestingly, we show that our BFT-STV protocol preserves the proportionality of STV while ensuring termination (Section 6.3.4).

- We implement this new protocol in a smart contract written in the Solidity programming language, making our protocol easily compatible and pluggable with smart contract supported blockchains [16, 39, 11, 1] (one can re-implement our protocol to make it work with a different smart contract programming language). Implementing the BFT-STV protocol on a smart contract comes with its own technical challenges. Smart contracts are public, thus to preserve the privacy of votes to avoid strategic voting, we employ a commit-reveal scheme (6.3.2).

- We prove that our protocol is correct (Section 6.3.4). In particular, we also show that BFT-STV satisfies proportionality without assuming synchrony. Our world-scale evaluations of BFT-STV with 200 validators of Ethereum-PoA and Smart Redbelly Blockchain [1] spanning 5 continents can elect 200 governors from 500 candidates with 1000 voters casting ballots within 6-12 minutes (Section 6.4).

- The BFT-STV smart contract alone is not sufficient to action the outcome of the election to reconfigure the blockchain. Thus, we present an automatic governance reconfiguration protocol as a secondary contribution that rotates governor sets periodically based on the BFT-STV smart contract election outcome. In particular, our protocol revokes the permissions of existing governors to elect new governors periodically.

Chapter Outline: The remainder of this chapter is as follows: First, we present our model (Section 6.1) followed by the proportional governance problem (Section 6.2). Next, we present our solutions to this problem in Sections 6.3 along with proof that our solution solves the proportional governance problem. In Section 6.4, we evaluate our solution show its feasibility. Section 6.5 presents a complementary automatic reconfiguration protocol that works with our

proportional governance solution. Section 6.6 discusses our contributions with other governance solutions. Finally, we conclude the chapter in Section 6.7.

6.1 Model

In this section, we present our computation model including the system model and threat model. The assumptions we make are encapsulated into these models.

6.1.1 System Model

We consider a distributed system of n governor nodes also known as a governor committee, identified by public keys I and network identifiers A (e.g., domain names or static IP addresses). We assume public key cryptography and that the adversary is computationally bounded. Hence, only the issuer of a transaction can *sign* it and any recipient can correctly verify the signature. Governor nodes (i) execute the consensus protocol to agree on a unique block to be appended to the chain and (ii) execute transactions and maintain a local copy of the state of the blockchain. Candidate nodes m are nodes eligible to become governors and are voted upon by n current governors to be included in new governor sets periodically. As we describe in Section 6.3.2, governors cannot vote for themselves due to a ballot verification. Proposing blocks to consensus provides governors with a block reward. As such, a candidate has an incentive to become a governor. We assume $m >> k$ s.t. k is the target next governor committee size. The number of Byzantine nodes in the candidate nodes set is assumed to be f_c s.t. $f_c \leq m/4$.

All candidates need to go through a KYC identification process, similar to the personal information requested from the Ethereum proof-of-authority network users before they can run a validator node [68]. For the initial set of governors to be sufficiently diverse, we can simply select governors from candidates based on their detailed information. A set of governors could then be selected depending on the provided information while ensuring multiple governors are not from the same jurisdiction, they are not employed by the same company, they represent various ethnicities, they are of balanced genders, etc.

As we target a secure blockchain system running over an open network like the Internet, we consider the Byzantine model [6], where nodes can fail arbitrarily by, for example, sending erroneous messages or delaying messages. We assume that a bound exists on the transmission delay of messages between nodes but it is not known a priori, a property called *partial synchrony* [47] (Chapter 2, Section 2.2).

6.1.2 Threat Model

As in previous blockchain works [71, 10, 30, 31], we assume a slowly adaptive adversary with a very limited bribing power that cannot bribe governors within a committee but can only bribe/corrupt up to f_c nodes between governance reconfigurations such that $f_c < m/4$ where m is the candidates.

As consensus cannot be solved with f Byzantine processes among n processes where $f \geq n/3$ and message delays are unknown [47], it is sufficient to bribe $n/3$ governors to lead correct

governors to disagree on the next block appended to the blockchain and thus create a fork in the blockchain. The attacker can then exploit this fork to have its transaction discarded by the system and then re-spend the assets he supposedly transferred in what is called a *double spend*. To mitigate such bribery attacks, we assume a governance reconfiguration protocol replaces existing governors with newly elected governors (Section 6.5). More specifically, once n new governors from m candidates are elected periodically (every X blocks) using the proportional governance, a governance reconfiguration occurs.

Due to the assumption of a slowly adaptive adversary that bribes/corrupts at most f_c candidate nodes s.t. $f_c < m/4$, a governance committee k ($k = n$) periodically elected proportionally from a diverse set m and reconfigured will have $f < n/3$ with high probability. As governors execute consensus and consensus cannot be solved in our model with $n/3$ or more Byzantine nodes [6], the assumption that at most f nodes are Byzantine s.t. $f < n/3$ with high probability is essential to solve consensus. This is a reasonable assumption made in prior work given that $m >> k$ [71, 31]. Within the governance committee period, this f will remain static as the slowly-adaptive adversary can only corrupt nodes between reconfigurations. Finally, we assume that a split in the blockchain does not occur leading to multiple elections for each instance of the blockchain.

Sybil Attacks

A Sybil attack consists of impersonating multiple identities to overwhelm a system—in the context of votes and voters, a Sybil attack could result in an adversary voting with multiple identities to alter the outcome of an election. In the context of candidates, a Sybil attack could result in an adversary assuming the identities of multiple candidates to alter the outcome of an election. Proof-of-stake-based voting approaches weigh a ballot cast by a voter based on the coins they have staked. Thus, minimizing the impact on the election outcome if an adversary splits its stake among multiple identities and cast ballots. However, PoS-based voting provides more opportunities to the wealthy, inadvertently creating an oligarchy.

We adopt a solution that does not weigh votes according to stake. As mentioned previously, the initial set of governors are elected from candidates. To receive permission to be a candidate, a node must authenticate using know-your-customer (KYC) data. Authentication through KYC copes with Sybil attacks by preventing the same authenticated user from assuming the identities of multiple candidates. It also prevents the current governors from voting with multiple identities.

6.2 The Proportional Governance Problem

Our goal is to solve the proportional governance problem to mitigate a governance oligarchy in blockchains. To put it simply, first, we offer blockchain governance that allows distributed users (i.e., current governors) to elect a committee of governors proportionally representative of the voters and without dictatorship, which solves the proportional governance problem (6.2.1). In this section, we define the proportional governance problem (6.2.1).

6.2.1 Proportional governance problem

We refer to the proportional governance problem as the problem of designing a BFT voting protocol in which n current governors rank m candidates to elect a committee of k new governors ($k < m$ and $m > n$) to ensure non-dictatorship as defined by Arrow [140] and proportionality as defined by Dummett [76], Woodland [79] and Elkind et al. [141]. The main distinction is that we adapt this problem from social choice theory to the context of distributed computing.

Definition 2 (The Proportional Governance Problem)**.** *The* secure governance problem *is for a distributed set of n current governors, among which $f \leq t < n/3$ are Byzantine (t is an upper bound on the number f of Byzantine governors, $f \leq t$), to elect a winning committee of k new governors among m candidates (i.e., $m > k$) such that the following two properties hold:*

- Proportionality: *if, for some whole numbers j, s, and k satisfying $0 < j \leq s \leq k$, more than $j(n-t)/(k+1)$ of voters put the same s candidates (not necessarily in the same order) as the top s candidates in their preference listings, then at least j of those s candidates should be elected.*
- Non-dictatorship: *a single adversary, controlling up to $f < n/3$ Byzantine voters (current governors), cannot always impose their individual preference as the election outcome.*

The need for these two properties stems from our goal of guaranteeing proportional representation (proportionality), but also disallowing a coalition of Byzantine nodes (i.e., an oligarchy) from imposing their decision on the rest of the system (non-dictatorship). Note that the non-dictatorship property differs slightly from the definition in [140] that did not consider a Byzantine coalition. In particular, our property considers coalitions and prevents them from imposing their preference in "all" cases.

6.3 Solution: Byzantine Fault Tolerant Proportional Governance

In this section, we present how to elect, despite f Byzantine nodes where $f \leq t < n/3$ (i.e., $t = n/3 - 1$), a diverse set of governors to mitigate the formation of an oligarchy. The idea is to allow a set of n blockchain nodes that are current governors to vote and elect the committee of next governors proportionally representing the current governor votes. To this end, we propose the *Byzantine Fault Tolerant Single Transferable Vote (BFT-STV)* smart contract that solves the proportional governance problem (Def. 2).

6.3.1 Overview

In order to guarantee that the election solves the proportional governance problem (Def. 2), we designed the BFT-STV algorithm and implemented it as a smart contract. In this section, we present the high-level pseudo code of the BFT-STV algorithm. To bootstrap, the initial permissions to vote are obtained by n initial governors after identification using KYC to ensure

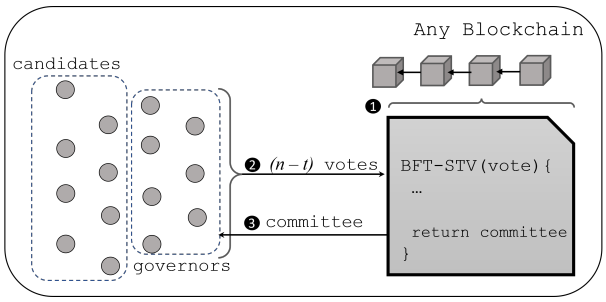

Figure 6.1: The smart contract that implements the BFT-STV protocol is on-chain ❶, takes as an input a set of at least $(n-t)$ ballots (each ranking k candidates among m) cast by $(n-t)$ voters among the n governors ❷ and outputs a committee of k elected nodes ❸ to play the role of the new governors. Note that the last committee of governors elected will then vote for the next committee of governors ❷ and so on (one can fix $k = n$ so that the committee size never changes).

diversity and prevent Sybil attacks (Section 6.1.2). Recall that governors cannot use the classic STV algorithm to elect a new committee as the smart contract has to progress despite up to t Byzantine voters not casting proper ballots and as the upper bound on the message delay is unpredictable. As depicted in Figure 6.1, the BFT-STV smart contract takes, instead, as an input $n-t$ ballots cast by the voters that are governors. Each ballot consists of a rank of all the candidates, hence the name *ordinal ballot*. Once the threshold $n-t$ of cast ballots is reached, the BFT-STV contract selects the governors based on the preference order indicated in the $n-t$ ballots. Traditionally, the STV algorithm consists of counting which candidates received a number of votes that exceed the quota $q_D = \frac{n}{k+1}$ where k is the size of the governance committee to be elected. However, as there can be at most t Byzantine nodes among the voters, we introduce the Byzantine quota $q_B = \frac{n-t}{k+1}$ (denoted q when clear from the context).

6.3.2 Byzantine Fault Tolerant Single Transferable Vote

Alg. 2 presents the main functions of the BFT-STV smart contract that the governors can invoke whereas Alg. 3 is the classic STV algorithm adapted to progress in a partially synchronous [47] environment and despite the presence of up to t Byzantine voters, hence its name STV_B.

Commit votes: Initially, the governors cast their hashed ballots by invoking the function $\mathsf{commitVote}(\cdot)$ at line 16 of Alg. 2. This prevents the ballot content of each voter from being known to other voters until the election counting begins. This is to mitigate strategic voting. Once governors cast $n-t$ private votes in the form of ballot hashes, the BFT-STV smart contract emits a broadcast notifying governors that their respective ballots can be revealed to commence counting votes (lines 21-23).

Reveal votes: Governors/voters upon receiving the broadcast in line 23, invoke the $\mathsf{reveal}(\cdot)$ function parsing the plain ballot b and the hash of this ballot h (line 24). If (1) the hash of

Algorithm 2 Byzantine Fault Tolerant Single Transferable Vote (BFT-STV) - Part 1

Initial state:
  k, the size of the targeted committee.
  n, the number of voters.
  t, an upper bound on the number f of byzantine replicas, $f \leq t$.
  m, the number of candidates per ballot.
  v, a mapping from candidates to their number of votes.
  ballots, the set of received ordinal ballots, initially empty
  C, the set of candidates.
  $E \subseteq C$, the set of eliminated candidates, initially empty.
  $S \subseteq C$, the set of winning candidates, initially empty.
  $pref[ballot] = index$, a map of ballot and its current preference index.
  $voted[vAddr] =$ false, a map of voter addresses and whether voted.
  $privateVotes[vAddr] = hashVote$, a map of voter addr. and priv. vote (ballot).
  countprivate, no. of private votes.
  $Sender[vAddr] = b$, a map of voter addr. and ballot.

commitVote(*hashVote*)**:**
  if $voted[vAddr] ==$ false **then** ▷ *prevents double voting*
    $privateVotes[vAddr] \leftarrow hashVote$ ▷ *store priv. votes for voters*
    $countprivate \leftarrow countprivate + 1$ ▷ *no. priv. votes received*
    $voted[vAddr] \leftarrow$ true ▷ *vAddr has voted*
    if $countprivate == n\text{-}t$ **then**
      votingEnded $\leftarrow$ true
      emit "threshold of votes reached" ▷ *notify all priv. votes are received*

reveal(b, h)**:**
  if *votingEnded* == true **then** ▷ *voters reveal votes if election ended*
    if hash(b) == h & $privateVotes[vAddr] == h$ **then** ▷ *hashes eq.*
      if *well-formed*(*b*) **then** $ballots \leftarrow ballots \cup \{b\}$
      if (*ballots* has $n - t$ ballots from distinct voters revealed) **then**
        change-committee(*ballots*) ▷ *replace committee*

change-committee(*ballots*)**:** ▷ *replace committee*
  for all $b \in ballots$ **do** ▷ *for each received ballot*
    if ($b[0] = c$ such that $c \in C$) **then**
      $v[c] \leftarrow v[c] + 1$ ▷ *number of 1st preferences = c*
    $pref[b] \leftarrow 0$ ▷ *assign pref. index of b to the first preference/index 0*
  $round \leftarrow 0$ ▷ *first round*
  while ($|S| < k$) **do** ▷ *until the new committee is full*
    $S \leftarrow \mathsf{STV}_\mathsf{B}(v, ballots, pref)$ ▷ *invoke classic STV*
    $round \leftarrow round + 1$ ▷ *increment round number*
    if ($|C| - |E| = k$) **then** break ▷ *stop eliminating*
  for all $b \in ballots$ **do** ▷ *for each ballot*
    for ($j = 0; j < m; j{+}{+}$) **do** ▷ *each candidate in decreasing pref. order*
      if ($|S| < k \wedge b[j] \in C \setminus S \setminus E$) **then** ▷ *if eligible*
        $S \leftarrow S \cup \{c\}$ ▷ *select c*
  emit S ▷ *explicitly emit committee*

Algorithm 3 Byzantine Fault Tolerant Single Transferable Vote (BFT-STV) - Part 2

Initial state:
k, the size of the targeted committee.
n, the number of voters.
t, an upper bound on the number f of byzantine replicas, $f \leq t$.
$q_B = \frac{n-t}{k+1}$, the quota of votes to elect a candidate.
C, the set of candidates.
$E \subseteq C$, the set of eliminated candidates, initially empty.
$S \subseteq C$, the set of winning candidates, initially empty.
$X \subseteq C$, the set of excess candidates, initially empty.

$\mathsf{STV}_B(v, ballots, pref)$**:**
if $\exists c \mid v[c] > q_B$ **then** ▷ *if the quota is exceeded*
  $S \leftarrow S \cup \{c\}$ ▷ *elect candidate*
  $X \leftarrow X \cup \{c\}$ ▷ *save candidates that exceed quota in X*
  $x[c] \leftarrow v[c] - q_B$ ▷ *excess vote from candidate c*
  for all $b \in ballots$ **do** ▷ *for each ballot*
    if $b[pref[b]] = c$ and $c \in X$ **then** ▷ *if current ballot pref = one of X*
      $count[c] \leftarrow count[c] + 1$ ▷ *the number of candidates c*
      $pref\text{-}next[b] \leftarrow pref[b] + 1$ ▷ *point to next preferred candidate*
      while $b[pref\text{-}next[b]] \in (S \vee E)$ **do** ▷ *while not uneligible*
        $pref\text{-}next[b] \leftarrow pref\text{-}next[b] + 1$ ▷ *try next pref. pointer*
      if $b[pref\text{-}next[b]] \notin (S \cup E)$ **then** ▷ *if eligible candidate found*
        $pref[b] = pref\text{-}next[b]$ ▷ *move the preference pointer*
        $z \leftarrow b[pref\text{-}next[b]]$ ▷ *next preferred candidate in ballot*
        $cand\text{-}next \leftarrow cand\text{-}next \cup \{\langle c, z\rangle\}$ ▷ *current&next candidates*
        $count[z] \leftarrow count[z] + 1$ ▷ *The number of candidates z*
  for all unique $\langle c, z\rangle \in cand\text{-}next$ **do** ▷ *transfer excess votes*
    $v[z] \leftarrow v[z] + x[c] \cdot (count[z]/count[c])$ ▷ *to next candidates*
if $\forall c : v[c] \leq q_B$ **then** ▷ *if no candidates exceed the quota in the round*
  $E \leftarrow (E \cup t \mid t = \mathsf{min}_{\forall c}(v[c]))$ ▷ *eliminate candidate with least votes*
  $transfer\text{-}vote \leftarrow v[t]$
  $v[t] \leftarrow 0$ ▷ *reset votes of least candidate to 0*
  for all $b \in ballots$ **do**
    while $s < size$ **do**
      if $b[s] = t$ **then** ▷ *store ballot and preference index...*
        $elimpointer \leftarrow elimpointer \cup (b, s)$ ▷ *...of least voted cand.*
      $s \leftarrow s + 1$ ▷ *Increment preference*
  for all $(b, s) \in elimpointer$ **do**
    if $b[s] = m \wedge m \in E$ **then** ▷ *If preference s of ballot b is eliminated*
      $pref\text{-}next[b] \leftarrow s + 1$
      $count[m] \leftarrow count[m] + 1$ ▷ *count of candidates m in all ballots*
      while $b[pref\text{-}next[b]] \in (S \vee E)$ **do** ▷ *until candidate is found*
        $pref\text{-}next[b] \leftarrow pref\text{-}next[b] + 1$ ▷ *...increment pref. pointer*
      if $b[pref\text{-}next[b]] \notin S \cup E$ **then**
        $pref[b] \leftarrow pref\text{-}next[b]$ ▷ *move the preference pointer*
        $z \leftarrow b[pref\text{-}next[b]]$
        $cand\text{-}next \leftarrow cand\text{-}next \cup (m, z)$ ▷ *least voted & next cand.*
        $count[z] \leftarrow count[z] + 1$ ▷ *the number of candidates z*
  for all unique $(m, z) \in cand\text{-}next$ **do** ▷ *transfer from least voted cand.*
    $v[z] \leftarrow v[z] + transfer\text{-}vote \cdot (count[z]/count[m])$
$X \leftarrow null$
return S ▷ *return the set of winning candidates*

b equals h and (2) h equals the hash of the ballot previously stored in the commit phase for the same voter, then the validity of the ballot b is checked. Upon successful validation, the ballot b is added to the list of *ballots* (lines 26-27). Note that verifying the validity of a ballot involves checking that the governors have not voted for themselves on their ballots and there are no duplicated preferences. Once the smart contract receives $n - t$ well-formed ballots, the change-committee$(\cdot)$ function is invoked (line 29).

Count votes: The change-committee$(\cdot)$ function starts by computing the score of the valid candidates as the number of votes they receive at lines 31–33. Valid candidates are initially selected through KYC (Section 6.1.2) before being periodically voted upon by governors to be elected as the next set of governors. A preference pointer is initialized to the first preference of each ballot at line 34. Then a new round of the STV election process starts (lines 35–38). This execution stops once the committee of new governors is elected (line 36). If before the targeted committee is elected, the number of eliminated candidates has reached a maximum and no more candidates can be eliminated to achieve the target committee size, then the STV election stops (line 39). The remaining non-eliminated candidates are elected by decreasing order of preferences at lines 40–43 until the target committee size is reached. Finally, the smart contract emits the committee of elected candidates (line 44), which notifies the replicas of the election outcome.

6.3.3 Classic STV with the Byzantine quota

Alg. 3 presents the classic STV algorithm but using the new Byzantine quota q_B by electing candidates whose number of votes exceeds q_B (line 55). This algorithm executes two subsequent phases: in the first phase (lines 54–71) the algorithm elects the candidates whose number of votes exceed the quota $q_B = \frac{n-t}{k+1}$; in the second phase (lines 72–94), the algorithm eliminates the least preferred candidate if no candidates received a number of votes that exceeds the quota. In each round of STV function call (line 37), when a candidate exceeds the quota (line 55), their excess votes are transferred to the next eligible preferences of the ballots that contain the candidate (line 71). In each round of ballot iteration, if no candidate has reached the quota, the candidate with the least votes is eliminated (line 73). This candidates' excess votes are transferred to the next eligible preference of the ballots that contain the candidate that received the least votes (line 93). The elimination of candidates stops when no more candidates can be eliminated to achieve the committee size (line 39). At this point, even though the remaining candidates did not receive enough votes to reach the quota, they are elected as part of the committee (line 43).

6.3.4 Proofs of Proportional Governance

In this section, we show that BFT-STV (Alg. 2 and 3) solves the proportional governance problem (Def. 2). To this end, the first theorem shows that the BFT-STV protocol ensures *Proportionality*. As mentioned in Section 6.2.1, recall that n, m, and k denote the number of voting governors, the number of candidates, and the targeted committee size, respectively. Note that the proof holds even if Byzantine voters vote in the worst possible way.

Theorem 5. *The BFT-STV multi-winner election protocol satisfies Proportionality.*

Proof. By examination of the code of Alg. 2 and 3, the only difference between BFT-STV and STV is the number of votes needed to elect a candidate. STV typically starts with n received ballots whereas the BFT-STV starts the election as soon as $(n-t)$ ballots are received (line 28 of Alg. 2), where t is the upper bound on the number f of Byzantine nodes and n is the total number of governors eligible to vote. This number of BFT-STV ballots is distributed among a larger number of candidates m. This can result in less than k candidates receiving enough votes to reach the classic STV quota where k is the size of the committee. By the Proportionality definition (Def. 6.2.1), we need to show that if $j \cdot (n-t)/(k+1)$ voters put the same s candidates as the top s candidates in their ballot preferences, then j of those s candidates will still be elected. The proof follows from [142, p. 48–49]: line 73 of Alg. 3 indicates that by elimination, and lines 40-43 of Alg. 2 indicates by electing the remaining non eliminated candidates in decreasing preference order, we elect the required k seats if k candidates cannot reach the q_B quota. Thus, we still elect the top j candidates such that $j = s = k$, satisfying proportionality.

The next theorem shows that the BFT-STV protocol ensures *Non-dictatorship* as defined in Def. 2.

Theorem 6. *The BFT-STV multi-winner election protocol satisfies Non-dicatorship.*

Proof. The proof shows the existence of an input of correct nodes for which a single adversary controlling f Byzantine nodes cannot always have its preference b_a be the winning committee. Let $b_a[-1]$ be the least preferred candidate of the adversary, we show that there exist preferences $b_1, ..., b_{n-f}$ from correct nodes such that the winning committee includes $b_a[-1]$. The result then follows from the assumption $k < m$.

By examination of the pseudocode, the winning committee is created only after receiving $n - t$ correctly formatted ballots (line 28 of Alg. 2). By assumption, there can only be at most $f \leq t < n/3$ ballots cast by Byzantine nodes. As a result, among all the $n - t$ received ballots, there are at least $n - 2t > n/3$ ballots cast from correct nodes. In any execution, an adversary controlling all the Byzantine nodes could have at most f ballots as the adversary cannot control the ballot cast by correct nodes. Let $b_1, ..., b_{n-f}$ be the ballots input by correct nodes to the protocol such that their first preference is the least preferred candidate of the adversary, i.e., $\forall i \in \{1, n-t\} : b_i = b_a[-1]$. Because $f \leq t < n/3$, we know that $b_a[-1]$ will gain more votes than any of the other candidates, and will thus be the first to be elected (line 55 of Alg. 3). By assumption, we have $k < m$, which means that there is a candidate the adversary prefers over $b_a[-1]$ that will not be part of the winning committee. Hence, this shows the existence of an execution where despite having an adversary controlling f Byzantine nodes, the adversary's preference is not in the winning governance committee.

6.4 Evaluation of Byzantine Fault Tolerant Proportional Governance

We evaluate our Byzantine Fault Tolerant Proportional Governance protocol on a world scale to observe its feasibility. To this end, we integrated our solution to Ethereum PoA and Smart Redbelly Blockchain (SRBB) [1] which are two smart contract supporting blockchains on the slower and faster end of the blockchain spectrum. We used the DIABLO blockchain benchmarking suite [28] that evaluates blockchains against defined workloads. Our defined workload consisted of 1000 voters (i.e., current governors) casting random ordinal ballots to 500 candidates to elect a committee of 200 governors using our BFT-STV smart contract. We deployed 200 AWS c5.2xlarge EC2 instances of Ethereum PoA and SRBB [1], spanning 10 AWS regions and 5 continents. Each AWS instance represented 5 governors of the respective blockchain realising a total of 1000 governors (i.e., 200×5), a restriction we placed due to budgetary constraints. Finally, we used a transaction sending rate of 1000 TPS, and considered the number of Byzantine voters as t=333 ($t < n/3$).

Table 6.2 depicts the time taken in seconds for the BFT-STV smart contract to elect a committee of 200 governors when 1000 voters (i.e., current governors) cast random ordinal ballots to 500 candidates on Ethereum PoA and SRBB. Ethereum PoA takes 728 seconds (i.e., 12 minutes) to elect a committee of 200 governors while SRBB [1] elected a committee of 200 governors within 358 seconds (i.e., 5.96 minutes).

Based on Table 6.2, the BFT-STV algorithm executed on a smart contract was able to elect a committee of governors within 12 minutes in one of the slowest smart contract supported blockchains (i.e., Ethereum) and within half that time on a faster blockchain (i.e., SRBB [1]).

Blockchain	#voters	#ballots	#candidates	#governors	time (seconds)
Ethereum PoA	1000	1000	500	200	728
SRBB	1000	1000	500	200	358

Table 6.2: 200 geo-distributed nodes of Ethereum PoA and SRBB representing 1000 voters (current governors) elects 200 new governors from 500 candidates using BFT-STV.

In the discussion (Section 6.6), we compare our governance election results against the reported results of other governance election methods. We especially consider the impacts on blockchain performance, availability, and security.

6.5 Automatic Governance Reconfiguration

In this section, we present a complementary governance reconfiguration protocol to elect governors periodically from the BFT-STV smart contract election outcome. The subsequent governance reconfiguration protocol assumes that all blockchain nodes in a network have the BFT-STV smart contract deployed at bootstrap time. Finally, We prove the correctness of our governance reconfiguration protocol (Def. 3).

Algorithm 4 Governance reconfiguration at a blockchain node

1: **initial:**
2: A is a set of IP addresses.
3: BC is a set of blockchains s.t. $Blockchain[start : end] \in BC$.
4: $Elected$: a set s.t. $S \in Elected$.
5: $count$: a map between a governor sets received and its occurrences.
6: $Bcount$: a map between a block and its occurrences in received prefixes.
7: S: newly elected governor committee
8: S_0: current governor committee
9:
10: **upon receiving** S, $Blockchain[start : end]$ **from a governor in** S_0**:** ▷ *recv. sc emits event from Alg.2, line 44 and bc prefix*
11: **if** $g \in S_0$ & $received[g] ==$ false **then** ▷ *prevents duplicate broadcast*
12: $Elected \leftarrow Elected \cup S$
13: $BC \leftarrow BC \cup Blockchain[start : end]$
14: **for all** $S \in Elected$ **do**
15: $count[S] \leftarrow count[S] + 1$
16: **if** $count[S] == n - t$ **then**
17: $threshold \leftarrow S$ ▷ *received same S from $n - t$*
18: **if** $threshold == S$ **then**
19: **for all** $bc \in BC$ **do**
20: **if** valid(bc) **then** ▷ *if prefix is valid, e.g., no duplicate blocks, etc*
21: **for all** $B \in bc$ **do**
22: $Bcount[B] \leftarrow Bcount[B] + 1$ ▷ *no. of block B in recv. prefix*
23: **if** $Bcount[B] == n - t$ & $B.index > highestIndex$ **then** ▷ *B is in $n - t$ gov. chains*
24: $highestIndex \leftarrow B.index$
25: $B_{decided} \leftarrow B$ ▷ *decided block so far*
26: **for all** $ip \in S$ **do** ▷ *for IPs in committee*
27: $A \leftarrow A \cup \{ip\}$ ▷ *add the IP address*
28: close-connect ▷ *stop connections with current governors*
29: connect(A) ▷ *connect with elected governors in A*
30: **if** $my\text{-}ip \in S$ **then** ▷ *if I'm an elected governor*
31: init$(B_{decided})$ ▷ *init. governor with $B_{decided}$ and its state.*
32: $threshold \leftarrow$ NULL ▷ *reset variables*
33: $Bcount \leftarrow$ NULL
34: $count \leftarrow$ NULL

6.5.1 Automatic Governance Reconfiguration Protocol

Alg. 4 allows switching from the current governance committee S_0 to the new governance committee S elected with the BFT-STV smart contract (Section 6.3). Note once a governor $g : g \in S_0$ emits S (line 44 of Alg. 2), they immediately stop processing any further blocks.

Once a blockchain node (i.e., candidate, governor, client) receives from governor $g : g \in S_0$ newly elected governors S and a blockchain prefix (line 10 of Alg. 4), the reconfiguration protocol commences. Note that duplicate broadcasts received by the same governor are not considered. First, every received S from a governor $g : g \in S_0$ is added to *Elected*. Thus, *Elected* stores all received S from current governors (line 12 of Alg. 4). Next, every blockchain prefix $Blockchain[start : end]$ received from a governor $g : g \in S_0$ is stored in BC (line 13 of Alg. 4). The blockchain prefix contains a chain of blocks where the start index *start* is the first block decided in the blockchain of governor g when in S_0 while the end index *end* is the last block decided in the blockchain of g when S was emitted by g.

Once a governor broadcast event S is received $n-t$ times (i.e., same S received $n-t$ times) from $n-t$ unique governors (line 17 of Alg. 4), that means at least $n-2t$ of the received S were from correct governors in the committee. Since $f \leq t < n/3$, S is the correct governor committee elected. When this condition is met, Alg. 4 executing on every blockchain node finds the block with the highest index decided by $n-t$ unique governors $g : g \in S_0$ using the blockchain prefixes received (lines 19-25 of Alg. 4).

Subsequently, the reconfiguration protocol closes the existing network connection with the previous governor committee (line 28 of Alg. 4). Then, every blockchain node connects with the new governor committee (line 29 of Alg. 4). Finally, if the blockchain node is also a governor elected in S, these governors initialize themselves with $B_{decided}$ which is the highest indexed block decided by $n-t$ governors from committee S_0.

For the sake of simplicity, we consider that nodes connect to the IP addresses of the new governors. The implementation could be easily adjusted so that nodes connect to a specific node ID that uniquely identifies a node. Since every blockchain node connects with the newly elected governor committee, (1) clients can send requests to the new governor committee (2) governors can reach consensus on governance decisions, and (3) governors can elect the next set of governors.

6.5.2 Proof of Correctness

Definition 3 (The Governance Reconfiguration Safety)**.** *The first block stored locally after governance reconfiguration by any two correct governor nodes in the governance committee is the same block.*

Any two correct governors in the same committee starting from the same block ensures that after the governance reconfiguration, governors start with the same block.

Theorem 7. *The governance reconfiguration (Alg. 4) satisfies the Governance Reconfiguration Safety property.*

Proof. By examination of Alg. 4 from the blockchain prefixes received from $n-t$ governors that sent S, each correct blockchain node finds the common block with the highest index $B_{decided}$ of all $n-t$ prefixes. This block is the highest confirmed/decided block by the governance committee S_0 (lines 19-25 of Alg. 4). If a correct local blockchain node is elected to the new governance committee S, then this node initializes with $B_{decided}$ (line 31 of Alg. 4). Every newly elected correct governor node in S initializes with the same $B_{decided}$.

6.6 Discussion

Based on our evaluation settings, the proportional governance solution can elect a new committee of governors within 6-12 minutes. Although a new governor committee can be elected within minutes, it is best for a blockchain to run governance elections and reconfigure after the current governors have been active for sometime. This is important to increase the availability of the blockchain and to lessen state downloads that can impact performance (candidates joining a governance committee should download the latest blockchain state). In fact, enterprise blockchains usually elect a governance committee every several hours. For example, Polkadot [54] and Tron [143] elect a committee of governors in 24 hours and 6 hours respectively. Governance election should strike an ideal balance between performance and blockchain security. Questions such as (1) how long is it safe to keep a governance committee static considering the risk of bribery? and (2) what are the service level guarantees in terms of blockchain availability and performance?, should be considered when defining the governance election frequency. In our blockchain proportional governance, to achieve stronger blockchain performance guarantees, the governors could collectively decide to delay executing an election and elect governors every 24 hours or 6 hours, similar to Polkadot or Tron [54, 143].

6.7 Summary

In this chapter, we presented *proportional governance* to mitigate the formation of an oligarchy of governors in blockchain governance committees. Proportional governance is the first solution that solves the proportional governance problem (Def 2). Our solution: (1) prevents an oligarchy among governors using proportionality (Def. 2), and (2) provides compatibility with a wide range of smart contract supported blockchains [11, 2, 144]. We proved that proportional governance ensures proportionality and non-dictatorship (Def. 2) and implemented proportional governance on Ethereum-PoA and Smart Redbelly Blockchain [1] which are two smart contract supporting blockchains. Our evaluation showed that our proportional governance solution implemented as BFT-STV on a smart contract (Alg. 2) can elect 200 governors within 6-12 minutes when 1000 voters cast ordinal ballots to 500 candidates.

Chapter 7

Conclusion

In this thesis, we have presented various contributions that pave the way to the overall improvement of Blockchain performance and security for DApp execution. As decentralized applications integrate further into society, the security and unprecedented pressure on blockchain infrastructure come into question. Current approaches to handle this pressure include techniques such as offloading the transaction processing off-chain (Layer 2). However, throughout this thesis, there have been alternative mechanisms identified that provide promising performance improvements and can be used in conjunction with these techniques. Our works presented on architecture decoupling and message redundancy show simple measures that can be implemented in various cases. Not only will these ideas increase the performance and overall usability of blockchains, but they also open doors for the further integration of DApps in daily life, further materialising the concept of Web3. Whilst we acknowledge that the work presented in this thesis is only a stepping stone towards the realisation of Web3, we believe our contributions have helped in improving the current state of the art.

In the remainder of this chapter, we summarize the achievements of our goals and objectives presented in Chapter 1. We then discuss future work and directions.

7.1 Goal and Objective Outcomes

The core research goal in this thesis was to enhance blockchain performance and security for DApp executions with the motivation of widening the adoption of Web3.

The first part of our core research goal was to enhance blockchain performance for DApp executions, which we identified as our first objective. Thus, we presented various contributions to enhance blockchain performance for DApp executions.

Smart Redbelly Blockchain: First, we investigated the reasons why modern blockchains cannot support realistic DApp workloads and found blockchain congestion to be the cause of performance degradation. Next, to reduce blockchain congestion and support real DApp workloads, we presented the provably secure Smart Redbelly Blockchain (SRBB). We empirically evaluated SRBB against Algorand, Avalanche, Diem, Solana, Quorum, and Ethereum. SRBB

not only committed all transactions of the demanding NASDAQ and Uber workloads, it comfortably outperformed 6 modern blockchains achieving an average throughput of 2000 TPS.

DApp-oriented Dynamic Transparent Sharding: To enhance blockchain performance for DApp executions, we then presented a DApp-oriented dynamic transparent sharding protocol that executed DApps concurrently in separate shards. Our sharding protocol was transparent in that any user could query the blockchain and know the current and past sharding configurations. The sharding protocol was dynamic in that it allowed the number of shards and the size of the shard to be adjusted at runtime. We evaluated our sharding protocol on SRBB to empirically show the boost in DApp performance. Our results showed that the sharded SRBB doubles the average and peak throughputs of SRBB while reducing the transaction losses when evaluated under an aggregated workload consisting of NASDAQ, Uber, and FIFA.

Collachain: To improve the performance of SRBB by other means than sharding, we presented a variant of SRBB coined Collachain that decouples the consensus and execution of SRBB. Collachain separated the consensus and execution into two separate machines and introduced RPC calls to facilitate communication between the two. Collachain improved the peak throughput of SRBB by 33%.

With the aforementioned contributions, we showed various novel methods to improve blockchain performance for DApp executions. The second part of our core research goal was to enhance blockchain security for DApp executions, which was our second objective.

Blockchain Proportional Governance: We presented a novel blockchain proportional governance protocol that mitigates an oligarchy of governors being formed in the blockchain. As an oligarchy can impede the secure execution of DApps, our blockchain proportional governance protocol enhanced blockchain security for DApp executions. The proportional governance protocol elected governors proportionally, mitigating an oligarchy in the election process. We proved the properties offered by our proportional governance protocol and empirically evaluated its performance. We showed that on a world-scale our blockchain proportional governance protocol can elect 200 governors within 6-12 minutes when 1000 voters cast ordinal ballots to 500 candidates.

7.2 Future Work

This thesis presented various contributions to improve the performance and security of blockchains to support the execution of DApps. While completely replacing the current version of the web with Web3 is an ongoing endeavour, our contributions have provided a foundation to build upon. As new protocols are developed to improve the performance and security of blockchains, it is likely that these would contribute towards further improving the performance and security of blockchains. While our thesis focused on the blockchain layer (*Layer 1*), *Layer 2* solutions appear to be widely adopted in the industry to boost the performance of blockchains [29,

145]. These *Layer 2* solutions, however, trades off transparency and decentralization for performance [29]. Ultimately, *Layer 2* solutions rely on the *Layer 1* to verify transaction execution proofs.

The works presented in this thesis open several avenues for future research directions. Below, a high-level overview of these pathways is described.

7.2.1 Mitigating Transaction Censorship

Our future work includes evaluating methods to mitigate transaction censorship in SRBB. As mentioned in Chapter 3 Section 3.6, we explore load balancing techniques to randomly forward each transaction from a client to different SRBB validators to mitigate censorship and increase the probability of transactions being executed. Alternatively, another possible method to mitigate transaction censorship is for a client to send a single transaction to $f + 1$ validators where at most f validators are Byzantine such that $f < n/3$. However, such an approach would require clients to know the validators prior to sending transactions, and, therefore, may not be ideal for a permissionless environment as validators can join and leave at will.

7.2.2 Performance Evaluations

The evaluations of SRBB, sharded SRBB, and Collachain was restricted to at most 200 geo-distributed c5.2xlarge AWS EC2 instances. This was particularly to ensure the evaluations were consistent and comparable with the Diablo evaluations in [28]. Potential future research can focus on evaluating the presented work at a much larger scale (e.g., 1000 AWS EC2 instances). Other aspects in the evaluation that could be looked into as a part of future research include (1) evaluating more DApp workloads and (2) using small instance sizes (e.g., AWS t2.micro) to observe SRBB's ability to perform on resource-constrained devices such as IoT (Internet-of-Things) devices.

7.2.3 A Unified Solution

While we presented various contributions to improve blockchain performance and security, we did not present a single overarching system that combines all the contributions and evaluates the potential of all these aspects working together. In other words, we first presented SRBB to improve blockchain performance for DApps and then used SRBB as a system to evaluate and compare the other protocols we introduced later. We did not incrementally implement all the new protocols we introduced in this thesis on SRBB to end up with a single overarching system that encapsulates all our contributions.

Future research directions include combining the contributions in this thesis to produce a single overarching system. More specifically, we intend to perform the following tasks in order (1) use the presented sharding protocol to produce sharded Collachain (2) implement the blockchain proportional governance on the sharded Collachain, and (3) evaluate the final system against state-of-the-art.

7.2.4 Integrating Layer 2 Solutions

As mentioned throughout this thesis, our work focused on the blockchain layer (Layer 1), and hence Layer 2 solutions were considered orthogonal to the contributions we presented. We believe that since Layer 2 eventually relies on the blockchain layer to include execution proofs, the performance and security of the blockchain layer was paramount. However, there is potential to integrate Layer 2 solutions with SRBB and Collachain presented in this thesis as a means to further improve the performance of DApps as a part of future work. In the path to realizing Web3, we believe there is no single path to success. Instead, appreciating existing state-of-the-art, encouraging developments, and integrating these in a secure way can go a long way in developing a truly decentralized web.

Glossary

Blockchain Consensus Reaching agreement on the execution order of transactions.

Blockchain Sharding Splitting blockchain tasks into separate groups of nodes known as shards.

DApps Decentralized Applications. These applications execute on the blockchain and inherit its decentralization.

Double spending The same coin is spent twice in a blockchain allowing an adversary to own more goods for the price of one coin.

Governance The processes followed to make decisions that modify the blockchain protocol.

Oligarchy A privileged group of people having control over a specific task.

PoS The process of staking assets in order to propose blocks. The probability of being selected to propose a block is proportional to the staked assets.

Smart Contract A piece of code that deterministically executes on the blockchain.

Solidity A Turing-complete smart contract language widely used to develop smart contracts.

Sybil Attacks An attack where an adversary assumes the identity of multiple users to overwhelm a system.

Web3 An new iteration of the web that is decentralized and services users through DApps executing on the blockchain instead of centralized web applications.

Zero-Knowledge Making a claim without revealing details about the claim.

List of Acronyms

SMR State Machine Replication

DApp Decentralized Application

BFT Byzantine Fault Tolerant

CFT Crash Fault Tolerant

RPC Remote Procedure Call

EVM Ethereum Virtual Machine

GST Global Stabilization Time

PoW Proof-of-Work

PoS Proof-of-Stake

PoA Proof-of-Authority

NFT Non Fungible Tokens

SRBB Smart Redbelly Blockchain

TVPR Transaction Validation and Propagation Reduction

RPM Reward Penalty Mechanism

TPS Transactions Per Second

KYC Know Your Customer

FPTP First-Past-The-Post

SNTV Single Non-Transferable Vote

STV Single Transferable Vote

IP Internet Protocol

Bibliography

[1] Deepal Tennakoon, Yiding Hua, and Vincent Gramoli. "Smart Redbelly Blockchain: Reducing Congestion for Web3". In: *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2023, pp. 940–950. DOI: 10.1109/IPDPS54959.2023.00098.

[2] Deepal Tennakoon and Vincent Gramoli. "Blockchain Proportional Governance Reconfiguration: Mitigating a Governance Oligarchy". In: *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 2023, pp. 545–556. DOI: 10.1109/CCGrid57682.2023.00057.

[3] Deepal Tennakoon and Vincent Gramoli. "Dynamic blockchain sharding". In: *5th International Symposium on Foundations and Applications of Blockchain 2022 (FAB 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2022.

[4] Deepal Tennakoon and Vincent Gramoli. "Transparent Sharding". In: *Data Engineering* (2022), p. 37.

[5] Deepal Tennakoon, Yiding Hua, and Vincent Gramoli. "CollaChain: A BFT collaborative middleware for decentralized applications". In: *arXiv preprint arXiv:2203.12323* (2022).

[6] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine generals problem". In: *Concurrency: the works of leslie lamport*. 2019, pp. 203–226.

[7] A Tutorial and F Schneider. "Implementing Fault-Tolerant Services Using the State-Machine Approach". In: *ACM Computing Surveys* 22.4 (1990), pp. 299–319.

[8] Tyler Crain et al. "Blockchain consensus". In: *ALGOTEL 2017-19èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*. 2017.

[9] Rihong Wang et al. "K-Bucket Based Raft-Like Consensus Algorithm for Permissioned Blockchain". In: *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*. 2019, pp. 996–999. DOI: 10.1109/ICPADS47876.2019.00152.

[10] Yossi Gilad et al. "Algorand: Scaling Byzantine Agreements for Cryptocurrencies". In: *Proceedings of the 26th Symposium on Operating Systems Principles*. SOSP '17. Shanghai, China: Association for Computing Machinery, 2017, 51–68. ISBN: 9781450350853. DOI: 10.1145/3132747.3132757. URL: https://doi.org/10.1145/3132747.3132757.

[11] JPMorgan Chase. *Quorum Whitepaper*. en. 2019. URL: https://github.com/ConsenSys/quorum/blob/master/docs/Quorum\%20Whitepaper\%20v0.2.pdf.

[12] Tyler Crain, Christopher Natoli, and Vincent Gramoli. "Red Belly: a Secure, Fair and Scalable Open Blockchain". In: *IEEE S&P*. May 2021, pp. 1501–1518.

[13] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. [Online] Available: https://bitcoin.org/bitcoin.pdf. 2008.

[14] Globenewswire. *Blockchain Market to Reach $403.36 Billion by 2030 - Exclusive Report by Meticulous Research*. [Online] Available: https://www.globenewswire.com/news-release/2023/06/06/2683010/0/en/Blockchain-Market-to-Reach-403-36-Billion-by-2030-Exclusive-Report-by-Meticulous-Research.html. 2023.

[15] Daniel Ruby. *75 Blockchain Statistics — Demographics, Cryptos& More (2023)*. [Online] Available: https://www.demandsage.com/blockchain-statistics. 2023.

[16] Gavin Wood et al. "Ethereum: A secure decentralised generalised transaction ledger". In: *Ethereum project yellow paper* 151.2014 (2014), pp. 1–32.

[17] Javad Zarrin et al. "Blockchain for decentralization of internet: prospects, trends, and challenges". In: *Cluster Computing* 24.4 (2021), pp. 2841–2866.

[18] Pedro Herrera. *2021 Dapp Industry Report*. [Online] Available: https://dappradar.com/blog/2021-dapp-industry-report. 2021.

[19] Andres Guadamuz. "These are not the apes you are looking for". In: *Communications of the ACM* 65.9 (2022), pp. 20–22.

[20] DAppRadar. *Top Blockchain Dapps*. [Online] Available: https://dappradar.com/rankings. 2023.

[21] Gavin Wood. *ĐApps: What Web 3.0 Looks Like*. Accessed: 2023-10-17, https://gavwood.com/dappsweb3.html. 2023.

[22] Frederik Stjernfelt et al. "Facebook and Google as offices of censorship". In: *Your post has been removed: tech giants and freedom of speech* (2020), pp. 139–172.

[23] Daniel Kreiss and Shannon C McGregor. "The "arbiters of what our voters see": Facebook and Google's struggle with policy, process, and enforcement around political advertising". In: *Political Communication* 36.4 (2019), pp. 499–522.

[24] Eduardo Hargreaves et al. "Biases in the Facebook News Feed: A Case Study on the Italian Elections". In: *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. 2018, pp. 806–812. DOI: 10.1109/ASONAM.2018.8508659.

[25] Wackerow. *Web2 vs Web3*. [Online] Available: https://ethereum.org/en/developers/docs/web2-vs-web3/. 2023.

[26] *ETH Network: So Congested Exchanges Are Forced to Disable ETH Wallets*. Accessed: 2023-02-24 - https://www.newsbtc.com/news/ethereum-network-congested-exchanges-forced-disable-eth-wallets/. 2022.

[27] *Solana Explains Reasons Behind the Recent Network Slowdown*. Accessed: 2023-02-24 - https://tinyurl.com/5f6xjbtp. 2022.

[28] Vincent Gramoli et al. "DIABLO: A Benchmark Suite for Blockchains". In: *18th European Conference on Computer Systems (EuroSys)*. 2023.

[29] Dave. *ZERO-KNOWLEDGE ROLLUPS*. [Online] Available: https://ethereum.org/en/developers/docs/scaling/zk-rollups. 2023.

[30] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. "RapidChain: Scaling Blockchain via Full Sharding". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS '18. Toronto, Canada: Association for Computing Machinery, 2018, 931–948. ISBN: 9781450356930. DOI: 10.1145/3243734.3243853. URL: https://doi.org/10.1145/3243734.3243853.

[31] Eleftherios Kokoris-Kogias et al. "OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding". In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pp. 583–598. DOI: 10.1109/SP.2018.000-5.

[32] G. Wang et al. "Prism Removes Consensus Bottleneck for Smart Contracts". In: *2020 Crypto Valley Conference on Blockchain Technology (CVCBT)*. 2020, pp. 68–77. DOI: 10.1109/CVCBT50464.2020.00011.

[33] Elli Androulaki et al. "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains". In: *Proceedings of the Thirteenth EuroSys Conference*. EuroSys '18. Porto, Portugal, 2018, 30:1–30:15. ISBN: 978-1-4503-5584-1.

[34] Rati Gelashvili et al. *Block-STM: Scaling Blockchain Execution by Turning Ordering Curse to a Performance Blessing*. 2022. DOI: 10.48550/ARXIV.2203.06871. URL: https://arxiv.org/abs/2203.06871.

[35] Louis Tremblay Thibault, Tom Sarry, and Abdelhakim Senhaji Hafid. "Blockchain scaling using rollups: A comprehensive survey". In: *IEEE Access* (2022).

[36] Tyler Crain et al. "DBFT: Efficient Leaderless Byzantine Consensus and its Application to Blockchains". In: *IEEE NCA*. 2018, pp. 1–8.

[37] Team Rocket. *Snowflake to Avalanche: A Novel Metastable Consensus Protocol Family for Cryptocurrencies*. Tech. rep. Accessed: 2021-12-01. 2018.

[38] The Diem Team. *DiemBFT v4: State Machine Replication in the Diem Blockchain*. https://developers.diem.com/papers/diem-consensus-state-machine-replication-in-the-diem-blockchain/2021-08-17.pdf. Aug. 2021.

[39] Anatoly Yakovenko. "Solana: A new architecture for a high performance blockchain v0. 8.13". In: *Whitepaper* (2018).

[40] Adem Efe Gencer, Robbert van Renesse, and Emin Gün Sirer. "Short Paper: Service-Oriented Sharding for Blockchains". In: *Financial Cryptography and Data Security*. Ed. by Aggelos Kiayias. 2017, pp. 393–401.

[41] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. "Parblockchain: Leveraging transaction parallelism in permissioned blockchain systems". In: *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2019, pp. 1337–1347.

[42] *Reject transaction sending a lot of transactions in a short period of time*. Accessed: 2020-09-09, https://github.com/cosmos/ethermint/issues/544.

[43] *Binance Smart Chain*. Accessed on 2022-07-21, https://github.com/bnb-chain/whitepaper/blob/master/WHITEPAPER.md. 2020.

[44] Casey Kuhlman et al. *Hyperledger Burrow (formerly eris-db)*. Accessed: 2020-11-14, https://www.hyperledger.org/wp-content/uploads/2017/06/HIP_Burrowv2.pdf. Mar. 2017.

[45] Jiaping Wang and Hao Wang. "Monoxide: Scale out Blockchains with Asynchronous Consensus Zones." In: *NSDI*. Vol. 2019. 2019, pp. 95–112.

[46] Christopher James Natoli. *The Road to El Diablo: Towards Secure High Performance Blockchains*. 2021.

[47] C. Dwork, N. Lynch, and L. Stockmeyer. "Consensus in the Presence of Partial Synchrony". In: *J. ACM* 35.2 (1988), pp.288–323.

[48] P. Ekparinya, V. Gramoli, and G. Jourjon. "Impact of Man-In-The-Middle Attacks on Ethereum". In: *Proc. 37th IEEE Int. Symp. Reliable Distrib. Syst. (SRDS)*. Oct. 2018, pp. 11–20.

[49] Achour Mostefaoui, Hamouma Moumen, and Michel Raynal. "Signature-Free Asynchronous Byzantine Consensus with t < n/3 and o(N2) Messages". In: New York, NY, USA: Association for Computing Machinery, 2014. ISBN: 9781450329446. DOI: 10.1145/2611462.2611468. URL: https://doi.org/10.1145/2611462.2611468.

[50] J.-P. Martin and L. Alvisi. "Fast Byzantine Consensus". In: *IEEE Transactions on Dependable and Secure Computing* 3.3 (2006), pp. 202–215. DOI: 10.1109/TDSC.2006.35.

[51] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. "The Bitcoin Backbone Protocol: Analysis and Applications". In: *EUROCRYPT*. 2015, pp. 281–310.

[52] Hyperledger Caliper. *Hyperledger Caliper: Blockchain performance benchmarking for Hyperledger Besu, Hyperledger Fabric, Ethereum and FISCO BCOS networks*. [Online] Available: https://hyperledger.github.io/caliper/. 2023.

[53] Andreas Krüeger. *Chainhammer: Ethereum benchmarking*. [Online] Available: https://github.com/drandreaskrueger/chainhammer. 2023.

[54] Jeff Burdges et al. *Overview of Polkadot and its Design Considerations*. Tech. rep. 2005.13456. arXiv, 2020.

[55] Lucianna Kiffer, Dave Levin, and Alan Mislove. "Stick a fork in it: Analyzing the Ethereum network partition". In: *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. 2017, pp. 94–100.

[56] Nick Webb. "A fork in the blockchain: income tax and the bitcoin/bitcoin cash hard fork". In: *North Carolina Journal of Law & Technology* 19.4 (2018), p. 283.

[57] *The ZILLIQA Technical Whitepaper*. https://docs.zilliqa.com/whitepaper.pdf. URL: https://docs.zilliqa.com/whitepaper.pdf (visited on 12/09/2020).

[58] George Kappos et al. “An empirical analysis of anonymity in zcash”. In: *27th {USENIX} security symposium ({USENIX} security 18)*. 2018, pp. 463–477.

[59] Peter Druschel, Frans Kaashoek, and Antony Rowstron. *Peer-to-Peer Systems: First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers*. Vol. 2429. Springer, 2003.

[60] Christopher Natoli and Vincent Gramoli. “The blockchain anomaly”. In: *2016 IEEE 15th international symposium on network computing and applications (NCA)*. IEEE. 2016, pp. 310–317.

[61] Arthur Gervais et al. “On the security and performance of proof of work blockchains”. In: *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 2016, pp. 3–16.

[62] Cong T. Nguyen et al. “Proof-of-Stake Consensus Mechanisms for Future Blockchain Networks: Fundamentals, Applications and Opportunities”. In: *IEEE Access* 7 (2019), pp. 85727–85745. DOI: 10.1109/ACCESS.2019.2925010.

[63] Robert Frank. *The wealthiest 10% of Americans own a record 89% of all U.S. stocks*. https://www.cnbc.com/2021/10/18/the-wealthiest-10percent-of-americans-own-a-record-89percent-of-all-us-stocks.html. URL: https://www.cnbc.com/2021/10/18/the-wealthiest-10percent-of-americans-own-a-record-89percent-of-all-us-stocks.html (visited on 10/18/2021).

[64] *EOS.IO Technical White Paper v2*. Accessed: 2020-12-07, https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md#consensus-algorithm-bft-dpos.

[65] *DPoS*. Accessed: 2022-12-07, https://tronprotocol.github.io/documentation-en/introduction/dpos/.

[66] *Guide: Delegated Proof of Stake (DPoS)*. Accessed: 2022-12-07, https://cardanofeed.com/guide-delegated-proof-of-stake-dpos-77202.

[67] *EOS Voting — Who is pulling the strings?* Accessed: 2022-12-07, https://medium.com/@_tlabs/eos-voting-who-is-pulling-the-strings-8a4944c45e61.

[68] Pavel Khahulin Igor Barinov Viktor Baranov. *POA Network White Paper*. https://github.com/poanetwork/wiki/wiki/POA-Network-Whitepaper. Sept. 2018.

[69] P. Ekparinya, V. Gramoli, and G. Jourjon. “The Attack of the Clones Against Proof-of-Authority”. In: *Community Ethereum Development Conference (EDCON’19)*. Apr. 2019.

[70] Joseph Bonneau. “Why buy when you can rent? Bribery attacks on bitcoin-style consensus”. In: *Financial Cryptography and Data Security: FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers 20*. Springer. 2016, pp. 19–26.

[71] Loi Luu et al. “A Secure Sharding Protocol For Open Blockchains”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’16. Vienna, Austria: Association for Computing Machinery, 2016, 17–30. ISBN: 9781450341394. DOI: 10.1145/2976749.2978389. URL: https://doi.org/10.1145/2976749.2978389.

[72] Svante Janson. *Phragmén’s and Thiele’s election methods*. Tech. rep. Technical report, 2016.

[73] Louis Kaplow and Steven Shavell. “Any non-welfarist method of policy assessment violates the Pareto principle”. In: *Journal of Political Economy* 109.2 (2001), pp. 281–286.

[74] Robert Frank. *The wealthiest 10% of Americans own a record 89% of all U.S. stocks*. https://www.cnbc.com/2021/10/18/the-wealthiest-10percent-of-americans-own-a-record-89percent-of-all-us-stocks.html. URL: https://www.cnbc.com/2021/10/18/the-wealthiest-10percent-of-americans-own-a-record-89percent-of-all-us-stocks.html (visited on 10/18/2021).

[75] D. Black. *The Theory of Committees and Elections*. Cambridge University Press, 1958.

[76] M. Dummett. *Voting Procedures*. Oxford University Press, 1984.

[77] Jonathan Lundell & I D Hill. “To advance the understanding of preferential voting system - Notes on the Droop quota”. In: *Voting matters* (2007).

[78] Nicolaus Tideman. “The Single Transferable Vote”. In: *Journal of Economic Perspectives* 9.1 (Mar. 1995), pp. 27–38. DOI: 10.1257/jep.9.1.27.

[79] Douglas Woodall. “Properties of preferential election rules”. In: *Voting Matters*. Accessed: 04/05/2021, https://www.votingmatters.org.uk/ISSUE3/P5.HTM. 1994.

[80] Piotr Faliszewski et al. *Multiwinner Voting: A NewChallenge for Social Choice Theory*. Lulu.com, 2017.

[81] Mustafa Al-Bassam et al. *Chainspace: A Sharded Smart Contracts Platform*. 2017. arXiv: 1708.03778 [cs.CR].

[82] *The ETH2 Upgrades*. Accessed: 2022-03-26. URL: https://ethereum.org/en/eth2/.

[83] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. “SharPer: Sharding Permissioned Blockchains Over Network Clusters”. In: *Proceedings of the 2021 International Conference on Management of Data*. New York, NY, USA: Association for Computing Machinery, 2021, 76–88. ISBN: 9781450383431. URL: https://doi.org/10.1145/3448016.3452807.

[84] Huan Chen and Yijie Wang. “SSChain: A full sharding protocol for public blockchain without data migration overhead”. In: *Pervasive and Mobile Computing* 59 (2019), p. 101055. ISSN: 1574-1192. DOI: https://doi.org/10.1016/j.pmcj.2019.101055. URL: https://www.sciencedirect.com/science/article/pii/S1574119218306370.

[85] C. Natoli and V. Gramoli. "The Balance Attack or Why Forkable Blockchains Are Ill-Suited for Consortium". In: *47th IEEE/IFIP Int. Conf. Dependable Syst. and Netw. (DSN)*. June 2017.

[86] Parinya Ekparinya, Vincent Gramoli, and Guillaume Jourjon. "The Attack of the Clones against Proof-of-Authority". In: *Proceedings of the Network and Distributed Systems Security Symposium (NDSS'20)*. Feb. 2020.

[87] Miguel Castro and Barbara Liskov. "Practical Byzantine Fault Tolerance and Proactive Recovery". In: *ACM Trans. Comput. Syst.* 20.4 (Sept. 2002), 398–461. ISSN: 0734-2071. DOI: 10.1145/571637.571640. URL: https://doi.org/10.1145/571637.571640.

[88] Ghassan O. Karame, Elli Androulaki, and Srdjan Capkun. "Double-Spending Fast Payments in Bitcoin". In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. CCS '12. Raleigh, North Carolina, USA: Association for Computing Machinery, 2012, 906–917. ISBN: 9781450316514. DOI: 10.1145/2382196.2382292. URL: https://doi.org/10.1145/2382196.2382292.

[89] Arthur Gervais et al. "On the Security and Performance of Proof of Work Blockchains". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. Vienna, Austria: Association for Computing Machinery, 2016, 3–16. ISBN: 9781450341394. DOI: 10.1145/2976749.2978341. URL: https://doi.org/10.1145/2976749.2978341.

[90] Y. Amir et al. "Scaling Byzantine Fault-Tolerant Replication toWide Area Networks". In: *International Conference on Dependable Systems and Networks (DSN'06)*. 2006, pp. 105–114. DOI: 10.1109/DSN.2006.63.

[91] Ittai Abraham et al. "Solida: A blockchain protocol based on reconfigurable byzantine consensus". In: *arXiv preprint arXiv:1612.02916* (2016).

[92] Eleftherios Kokoris Kogias et al. "Enhancing bitcoin security and performance with strong consistency via collective signing". In: USENIX Association. 2016.

[93] Miya Chen Yu-Te Lin markya0616 and bailantaotao. *Istanbul Byzantine Fault Tolerance*. https://github.com/ethereum/EIPs/issues/650. June 2017.

[94] Roberto Saltini and David Hyland-Wood. "IBFT 2.0: A safe and live variation of the IBFT blockchain consensus protocol for eventually synchronous networks". In: *arXiv preprint arXiv:1909.10194* (2019).

[95] Maofan Yin et al. "HotStuff: BFT Consensus with Linearity and Responsiveness". In: *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. PODC '19. Toronto ON, Canada: Association for Computing Machinery, 2019, 347–356. ISBN: 9781450362177. DOI: 10.1145/3293611.3331591. URL: https://doi.org/10.1145/3293611.3331591.

[96] Maofan Yin et al. "HotStuff: BFT consensus with linearity and responsiveness". In: *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. 2019, pp. 347–356.

[97] Guy Golan Gueta et al. "SBFT: a Scalable and Decentralized Trust Infrastructure". In: *Proceedings of the 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2019.

[98] VMware. *Concord*. Accessed: 2020-11-28, https://github.com/vmware/concord.

[99] John R Douceur. "The sybil attack". In: *International workshop on peer-to-peer systems*. Springer. 2002, pp. 251–260.

[100] Mike Isaac and Sheera Frenkel. *Facebook Security Breach Exposes Accounts of 50 Million Users*. Accessed: 2023-02-24. Sept. 2018. URL: https://www.nytimes.com/2018/09/28/technology/facebook-hack-data-breach.html.

[101] Jeremias Prassl. *Humans as a Service: The Promise and Perils of Work in the Gig Economy*. Oxford Press, 2018. ISBN: 9780198797012. DOI: 10.1093/oso/9780198797012.001.0001.

[102] Brianna Provenzano. *YouTube Is Down For Everyone Right Now [Update: It's Back]*. Accessed: 2020-11-14. Nov. 2020. URL: https://www.msn.com/en-us/money/other/youtube-is-down-for-everyone-right-now-update-it-s-back/ar-BB1aVtNh.

[103] Michael Spain, Sean Foley, and Vincent Gramoli. "The Impact of Ethereum Throughput and Fees on Transaction Latency During ICOs". In: *Proceedings of the International Conference on Blockchain Economics, Security and Protocols (Tokenomics)*. Vol. 71. OASIcs.

[104] Gustavo A Oliva, Ahmed E Hassan, and Zhen Ming Jack Jiang. "An exploratory study of smart contracts in the Ethereum blockchain platform". In: *Empirical Software Engineering* 25.3 (2020), pp. 1864–1904.

[105] *Ethermint*. Accessed: 2020-11-14, https://github.com/cosmos/ethermint.

[106] Laszlo Dobos. *USDC blacklist cost users an extra 3.6 million–per month*. 2022.

[107] Ye Wang et al. "Impact and User Perception of Sandwich Attacks in the DeFi Ecosystem". In: *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 2022.

[108] *Introduction to Staking*. Accessed: 2022-12-22, https://wiki.polkadot.network/docs/learn-staking. 2022.

[109] Gabriel Bracha. "Asynchronous Byzantine agreement protocols". In: *Information and Computation* 75.2 (1987), pp. 130–143.

[110] Ittai Abraham et al. "Solidus: An incentive-compatible cryptocurrency based on permissionless byzantine consensus". In: *CoRR, abs/1612.02916* (2016).

[111] *Slashing*. Accessed: 2022-08-09, https://wiki.polkadot.network/docs/learn-staking#slashing.

[112] *Slashing*. Accessed: 2022-08-09, https://consensys.net/blog/codefi/rewards-and-penalties-on-ethereum-20-phase-0/.

[113] Alejandro Ranchal-Pedrosa and Vincent Gramoli. "TRAP: The Bait of Rational Players to Solve Byzantine Consensus". In: *ASIACCS*. 2022, 168–181.

[114] *The Lifecycle of an Operation in Tezos*. Accessed on 2022-07-21, https://medium.com/everstake/how-does-slashing-work-in-tezos-and-why-is-it-important-to-delegate-only-to-reliable-bakers-like-a6c931e93c56. 2019.

[115] M. Toulouse, H. K. Dai, and Q. L. Nguyen. "A Consensus-Based Load-Balancing Algorithm for Sharded Blockchains". In: *Future Data and Security Engineering*. Springer, 2021, pp. 239–259.

[116] SJ Wels. "Guaranteed-TX: The exploration of a guaranteed cross-shard transaction execution protocol for Ethereum 2.0." MA thesis. University of Twente, 2019.

[117] Jelle Hellings et al. "Cerberus: Minimalistic multi-shard byzantine-resilient transaction processing". In: *arXiv preprint arXiv:2008.04450* (2020).

[118] Jae Kwon and Ethan Buchman. *Cosmos White Paper*. Accessed: 2021-25-03. URL: https://v1.cosmos.network/resources/whitepaper.

[119] Jianting Zhang et al. "SkyChain: A Deep Reinforcement Learning-Empowered Dynamic Blockchain Sharding System". In: *Proceedings of the 49th International Conference on Parallel Processing*. ICPP '20. New York, NY, USA: Association for Computing Machinery, 2020. ISBN: 9781450388160. DOI: 10.1145/3404397.3404460. URL: https://doi-org.ezproxy.library.sydney.edu.au/10.1145/3404397.3404460.

[120] Hung Dang et al. "Towards Scaling Blockchain Systems via Sharding". In: *Proceedings of the 2019 International Conference on Management of Data*. SIGMOD '19. Amsterdam, Netherlands: Association for Computing Machinery, 2019, 123–140. ISBN: 9781450356435. DOI: 10.1145/3299869.3319889. URL: https://doi-org.ezproxy.library.sydney.edu.au/10.1145/3299869.3319889.

[121] Sajjad Rahnama et al. "RingBFT: Resilient consensus over sharded ring topology". In: *arXiv preprint arXiv:2107.13047* (2021).

[122] E. Fynn, A. Bessani, and F. Pedone. "Smart Contracts on the Move". In: *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2020, pp. 233–244. DOI: 10.1109/DSN48063.2020.00040.

[123] Cosmos Networks. *Map Of Zones*. Accessed: 2022-03-10. URL: https://mapofzones.com/?testnet=false&period=24&tableOrderBy=ibcVolume&tableOrderSort=desc.

[124] Russell Brandom. *$1.7 million in NFTs stolen in apparent phishing attack on OpenSea users*. Accessed: 2022-03-11. URL: https://www.theverge.com/2022/2/20/22943228/opensea-phishing-hack-smart-contract-bug-stolen-nft.

[125] randao.org. *Randao: Verifiable Random Number Generation*. Tech. rep. Accessed February 2022. randao.org, 2017. URL: https://www.randao.org/whitepaper/Randao_v0.85_en.pdf.

[126] Luciano Freitas de Souza et al. "RandSolomon: optimally resilient multi-party random number generation protocol". In: *arXiv preprint arXiv:2109.04911* (2021).

[127] Finck Michelle. “Blockchain Governance”. In: *Blockchain Regulation and Governance in Europe*. Cambridge University Press, 2018, pp. 182–209. DOI: 10.1017/9781108609708.007.

[128] Vlad Zamfir. “Blockchain governance”. In: *Ethereum Community Conference*. Accessed: 2021-05-28, https://www.youtube.com/watch?v=PKyk5DnmW50. 2019.

[129] *BLOCK PRODUCERS RANKING - REAL TIME STATISTICS*. Accessed: 2020-11-14, https://eosauthority.com/producers_rank.

[130] Lucianna Kiffer, Dave Levin, and Alan Mislove. “Stick a Fork in It: Analyzing the Ethereum Network Partition”. In: *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. 2017, pp. 94–100.

[131] Nick Webb. “A Fork in the Blockchain: Income Tax and the Bitcoin/Bitcoin Cash Hard Fork”. In: *North Carolina Journal of Law & Technology* 19.4 (2018).

[132] Robert Dixon. “Fair criteria and procedures for establishing legislative districts”. In: *Policy Studies Journal* 9.6 (1981), p. 839.

[133] *Proportional Representation Voting Systems of Australia’s Parliaments*. Accessed:2021/06/04 –https://www.ecanz.gov.au/electoral-systems/proportional. 2021.

[134] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. “Reaching Agreement in the Presence of Faults”. In: *J. ACM* 27.2 (1980), pp. 228–234.

[135] Kwon J. *Tendermint: Consensus without Mining*. 2014.

[136] Rafael Pass and Elaine Shi. “Hybrid consensus: Efficient consensus in the permissionless model”. In: *31st International Symposium on Distributed Computing (DISC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2017.

[137] Guillaume Vizier and Vincent Gramoli. “ComChain: A Blockchain with Byzantine Fault Tolerant Reconfiguration”. In: *Concurrency and Computation, Practice and Experience* 32.12 (Oct. 2019).

[138] Shehar Bano et al. *State Machine Replication in the Libra Blockchain*. Accessed: 2019-10-01, https://developers.libra.org/docs/assets/papers/libra-consensus-state-machine-replication-in-the-libra-blockchain.pdf. 2019.

[139] Alysson Bessani et al. “From Byzantine Replication to Blockchain: Consensus is Only the Beginning”. In: *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 2020, pp. 424–436.

[140] Kenneth J. Arrow. “A Difficulty in the Concept of Social Welfare”. In: *Journal of Political Economy* 58.4 (1950), pp. 328–346.

[141] Edith Elkind et al. “Properties of multiwinner voting rules”. In: *Social Choice and Welfare* 48.3 (2017), pp. 599–632.

[142] Svante Janson. “Thresholds quantifying proportionality criteria for election methods”. In: *arXiv preprint arXiv:1810.06377* (2018).

[143] Decrypt Staff. *Tron Governance: How to Vote Using TRX*. Accessed:2022-10-14, https://decrypt.co/resources/tron-governance-how-to-vote-using-trx.

[144] *Ethereum Proof-of-Authority Consortium - Azure*. en-us. https://docs.microsoft.com/en-us/azure/blockchain/templates/ethereum-poa-deployment. URL: https://docs.microsoft.com/en-us/azure/blockchain/templates/ethereum-poa-deployment (visited on 09/09/2019).

[145] *Bring Ethereum to everyone*. [Online] Available: https://polygon.technology/polygon-zkevm. 2023.